НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій, СТІРЕНКО

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Інформаційна платформа для пошуку подій за інтересами (клієнтська частина)»

Виконав (-ла):
студент (-ка) IV курсу, групи ІП-64
Соляніков Павло Олександрович _____

Керівник:
Асистент кафедри ОТ
Каплунов Артем Володимирович _____

Консультант з нормконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____

Рецензент:

_____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6424. 00.001 ВП	Відомість дипломного проєкту	1	
3	А4	ДП 6424. 00.002 ТЗ	Технічне завдання	3	
4	А4	ДП 6424. 00.003 ПЗ	Пояснювальна записка		
5	А4	ДП 6424. 00.004 Д1	Загальна схема додатку	1	
6	А4	ДП 6424. 00.005 Д2	Принципова схема	1	
5	А4	ДП 6424. 00.006 Д3	Структурна схема	1	
6	А4	ДП 6424. 00.007 Д4	Діаграма класів сутності Подія	1	

				ДП 6424 00.001		
	ПІБ	Підп.	Дата			
Розробн.	Соляніков П.О.			Відомість дипломного проєкту	Лист	Листів
Керівн.	Каплунов А.В.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІП-64	
Н/контр.	Сімоненко В.П.					
Зав.каф.	Стіренко С.Г.					

Національний технічний університет України
«Київський політехнічний інститут»

Інститут (факультет) інформатики та обчислювальної техніки
(повна назва)

Кафедра обчислювальної техніки
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 121 «Інженерія програмного забезпечення»
(повна назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____Сергій, СТІРЕНКО
"___" _____ 2020 року

ЗАВДАННЯ
на дипломний проєкт студента

_____Павло СОЛЯНІКОВ_____
(ім'я, прізвище)

1. Тема проєкту *«Інформаційна платформа для пошуку подій за інтересами (клієнтська частина)»*

керівник проєкту _____ *Асистент кафедри ОТ Артем КАПЛУНОВ*_____,
(ім'я, прізвище, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "___" _____ 20__ року N ___

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту __технічна документація, теоретичні дані, інтернет-публікації за темою роботи _____

4. Зміст пояснювальної записки

− Провести огляд середовища розробки Xcode та методи праці з IOS Core;
− Провести аналіз засобів для розробки програмного забезпечення;
− Розробити програмну реалізацію взаємодії з BackEnd, кластеризації за допомогою MapKit ;
− Розробити архітектуру додатку та аналіз розробленої системи;

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	*Асистент кафедри ОТ Каплунов А. В.*	*10.03.2020*	*31.05.2020*
2	*Асистент кафедри ОТ Каплунов А. В.*	*10.03.2020*	*31.05.2020*
3	*Асистент кафедри ОТ Каплунов А. В.*	*10.03.2020*	*31.05.2020*
4	*Асистент кафедри ОТ Каплунов А. В.*	*10.03.2020*	*31.05.2020*

7. Дата видачі завдання *15.12.2019*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	*Затвердження теми роботи*	*15.12.2019*	
2	*Вивчення та аналіз завдання*	*20.12.2019-02.02.2020*	
3	*Проведення огляду середовища розробки Xcode та метоіви праці з IOS Core*	*02.02.2020-01.03.2020*	
4	*Проведення аналізу засобів для розробки програмного забезпечення*	*01.03.2020-10.03.2020*	
5	*Розроблення програмної реалізації взаємодії з BackEnd, кластеризації за допомогою MapKit*	*10.03.2020-12.04.2020*	
6	*Розроблення архітектури додатку та аналіз розробленої системи*	*12.04.2020-18.05.2020*	
7	*Оформлення матеріалів роботи*	*18.05.2020-31.05.2020*	
8	*Передзахист*		
9	*Захист*		

Студент _____ Соляніков П. О
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ Каплунов А. В
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У роботі був розроблений додаток на платформі Apple IOS. Додатком є соціальна мережа для пошуку подій. Останнім часом люди почали все більше віддавати перевагу мобільним додаткам замість веб-сторінок. Мобільні додатки значно зручніші та швидші. Сьогодні майже кожен користується соціальними мережами. Але майже всі слугують для віртуального спілкування. Наслідками є відсутність живого спілкування, замкненість, підміна реальності. Тому виникає необхідність у додатку, який буде стимулювати користувачів на живе спілкування. Запропонований додаток дає можливість користувачам створювати власні події та ділитися ними з усіма. Події можуть бути абсолютно різного характеру. Починаючи з запрошення приєднатися до гаражного рок-клубу, чи гуртку юних любителів робототехніки, закінчуючи подією про набір футбольної команди для гри між містами. У порівнянні із наявними аналогами додатку основними перевагами розробленого додатку є швидкий перегляд події на карті, можливість підписатися на конкретного користувача і стежити за його подіями, виставлення подіє є безкоштовною, тобто кожен має можливість знайти однодумців та проводити час разом.

ABSTRACT

The application was developed on the Apple IOS platform. The application is a social network for finding events. Recently, people have started to prefer mobile applications instead of web pages. Mobile applications are much more convenient and faster. Today, almost everyone uses social networks. But almost all are used for virtual communication. The consequences are the lack of live communication, isolation, substitution of reality. Therefore, there is a need for an application that will encourage users to communicate live. The proposed application allows users to create their own events and share them with everyone. Events can be completely different. From an invitation to join a garage rock club or a group of young

robotics enthusiasts, to an event about recruiting a football team to play between cities. Compared to the available analogues of the application, the main advantages of the developed application are a quick view of the event on the map, the ability to subscribe to a specific user and follow his events, the event is free, i.e everyone has the opportunity to find like-minded people and spend time together.

Технічне завдання до дипломного проекту
ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку додатку соціальної мережі для пошуку подій. Область застосування: перегляд актуальних подій у своїй місцевості, створення своїх власних подій, пошук однодумців.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки э завдання на виконання розробки клієнтської частини інформаційної платформи для пошуку подій, затвердженою кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є виконання розробки клієнтської частини інформаційної платформи для пошуку подій.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки є відкриті джерела з простору Інтернет науково-технічні статі, пересічні теми та технічна література.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Розробка інтерфейсу для користувача на основі мобільної платформи IOS.

5.2. Вимоги до програмного забезпечення для мобільного додатку:

- Операційна система IOS

5.3. Вимоги до апаратного забезпечення

- Наявність версії OS 11 і вище
- Процесор моделі A10 і вище

Пояснювальна записка
до дипломного проєкту
на тему: «Інформаційна платформа для пошуку подій за інтересами (клієнтська частина)»

Київ – 2020 року

ЗМІСТ

ВСТУП

В бакалаврській дипломній роботі було запропоновано та реалізовано програмний продукт формату соціальної мережі для пошуку подій за інтересами . Програма дозволяє дивитися актуальні події у своїй місцевості, а також створювати свої власні події, які будуть видні іншим користувачам. Програмний продукт було створено для продуктів Apple IPhone. Реалізація на мові Swift.

Актуальність теми в цілому обумовлена тим, що люди проводять дуже багато часу у мережі Інтернет. Кожен може знайти дудь-яку інформацію за лічені хвилини. Соціальні мережі стрімко розвиваються, а до них залучається все більше користувачів.

Практичним значенням реалізованого додатку є можливість користувачів знаходити однодумців, розвиватися, вчитися та розважатися разом.

Об'єктом даного дослідження є методи написання додатку на платформі IOS. Предметом даної роботи є створення гнучкої архітектури додатку щоб можна було швидко пристосовуватися до аудиторії, яка буде користуватися ним.

Метою роботи полягає у систематизації знань про розробку мобільних додатків щодо вирішення поставленої задачі.

Задачі, що поставлення для досягнення мети даної роботи:

- Розглянути основні методи, що використовуються у створенні мобільних додатків на базі IOS.
- Створити гнучку архітектуру додатку.
- Проаналізувати розроблений додаток.

РОЗДІЛ 1.
ОСНОВНІ ВИЗНАЧЕННЯ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Соціальна мережа

Соціальна мережа - багатокористувацький веб-сайт, контент якого наповнюється безпосередньо учасниками мережі. Сайт являє собою соціальне середовище, у якій групи користувачів, об'єднані спільними інтересами, можуть спілкуватися. Зв'язок здійснюється за допомогою миттєвого обміну повідомленнями. Також існують соціальні мережі для прослуховування музики, перегляду відео-контену та інші.

Важко знайти галузь людської діяльності, яка не була б представлена в Інтернеті. Інтернет - найбільший в світі джерело розваг. Ігри, музика, театр, кіно – все це вже давно в Інтернеті та користується чималим попитом.

1.2 Перші соціальні мережі

Приблизно в середині 90-х років з'явилися перші соціальні мережі. Вони надавали можливість користувачам спілкуватися один з одним, подібно до сервісів : ICQ, eGroups / OneList, Evite. Такі мережеві сервіси не були соціальними мережами, але, тим не менш, вони стали тими сервісами, я яких почався розвиток спілкування і взаємодії користувачів. Останнім часом соціальні мережі стають все більші і більші. Отже, вплив індустрії соціальних мереж на сучасне життя незаперечно [1].

Профіль учасника – найголовніший атрибут соціальної мережі. Профіль зазвичай містить ім’я користувача, його вік, сімейний стан, освіту, роботу, інтереси.

Завдяки профілю – аккаунт користувача зможуть шукати та знаходити інші учасники. Також одна з типових рис соціальних мереж - система «друзів» і «груп»

1.3 Найбільші соціальні мережі

Поява соціальних мереж, таких як LinkedIn або Flickr та таких просторів, як Digg, WordPress або YouTube чималий ріст інтересу до соціальних мереж. У 2004 році була створена одна з наймасштабніших соціальних мереж -

Facebook. Ця нова соціальна мережа не зайняла багато часу, щоб подолати MySpace, як лідера щодо щомісячних відвідувачів. Такий її успіх, що My Space повинен був змінити свою стратегію і, нарешті, створив новий дизайн, представляючи себе як розважальну платформу, а не соціальну мережу. Facebook, створений Марком Цукербергом, зародився спочатку з метою підключення студентів університету Гарварду. Фактично більше половини студентів університету підписалися на його у перший місяць роботи. Через рік після його запуску Facebook був у 500 американських університетах, де було понад 2 мільйони користувачів. У 2006 році Facebook став сервісом, відкритим для всіх. Що повністю змінило історію соціальних мереж [2]. В даний час ця соціальна мережа визнана однією з найважливіших усіх часів. Насправді, це все-таки соціальна мережа з найбільшою кількістю користувачів: 1800 мільйонів активних щомісячних користувачів. В останні роки з'являється все більше соціальних мереж, які з'являються в аудиторіях з різними характеристиками. Деякі з соціальних мереж за останні п'ять років:

- Pinterest. Це платформа для обміну зображеннями між користувачами.
- Tumblr. Це платформа для міні-блогів, яка дозволяє користувачам публікувати тексти, відео, зображення, посилання або цитати.
- Youtube. Веб-сайт, на якому користувачі можуть завантажувати та ділитися вмістом у вигляді відео. У ньому розміщені мільйони музичних відео, а також телебачення або відеоблоги.
- LinkedIn. Це більш спеціалізована соціальна мережа, спрямована на компанії та підприємства. Кожен користувач може створити профіль, де вільно виставляє свій досвід роботи та якості. Таким чином соціальна мережа ставить мільйони професіоналів у контакт один з одним.
- Snapchat. Цей мобільний додаток дозволяє надсилати файли, такі як зображення чи відео, які зникають з мобільного пристрою одержувача

протягом однієї-десяти секунд після його перегляду. Весь вміст подається через приватні повідомлення.

1.4 Розбір існуючих сервісів

1.4.1 Facebook Local

Додаток - це спосіб дізнатися, що відбувається незабаром і поблизу. Він поєднує в собі речі, які Facebook завжди робив добре - інформацію про місця, рекомендації від ваших друзів, цікаві речі, що відбуваються у вашому місті, і розміщує його в одному місці.

Додаток Local складається з трьох розділів.

На головному екрані пропонуються швидкі пошуки ресторанів чи напоїв, а також перелік подій, що відбуваються неподалік, які можуть вас зацікавити.

Ви також можете побачити, до яких подій висловили інтерес ваші друзі, якщо ви хочете позначити їх.

Другий розділ - це лише карта та пошукова система. Ви можете шукати "каву" або "тайську їжу", сортувати за "відкрити зараз", як і Yelp. Короткий погляд, це не настільки повна база даних, як Yelp, а також не має стільки оглядів чи оцінок, але це досить корисний інструмент. Там, де Facebook починає відрізнятись, це дозволяє вам шукати "комедію" та "літературу" та сортувати за "пізньою ніччю", щоб знайти що робити після обіду.

Третя вкладка - це лише ваш календар, спочатку насичений подіями Facebook, на які ви відповіли, але також включає будь-які календарі, до яких ви хочете надати їм доступ. Насправді це приголомшливо приємний календарний додаток.

Нова заява місії Facebook: "Дати людям владу будувати громаду та наближати світ". Компанія намагається знайти способи повернути людей у реальний світ. І всі ці функції також повернуться до головного додатку Facebook. Але Facebook спробував це раніше: він запустив додаток Events рік тому, і фактично досяг нульового успіху.

Події були лише способом побачити, що ваші друзі у Facebook роблять без вас. Local - це повний путівник, де б ви не були. Якщо додаток вдасться,

легко уявити безліч інших функцій Facebook, зручних для місцевого використання та бізнесу, які також надходять до Local. Можливо, одного дня ви зможете замовити їжу на Local або придбати квитки на захід, який вас хвилює. Ви, безумовно, зможете спілкуватися з підприємствами або, принаймні, чат-ботами, що намагаються Facebook створити бізнес. Додаток може переконати людей вкласти більше енергії на свої сторінки у Facebook, зберегти кращий календар подій та використовувати спільний доступ до локації (та Історії), щоб допомогти людям знайти один одного та побачити, що відбувається. Події ніколи не були платформою, гідною власного простору, але Local може бути [3].

1. Перегляд останніх дій, заходів і місць, з якими взаємодіють ваші друзі, а також перегляд новин від організаторів заходів і сторінок, на які ви підписані.
2. Пошук заходів і дій поруч з вами на інтерактивній карті. Можливість фільтрації за часом, категорії, місця і так далі.
3. Отримання рекомендацій з урахуванням того, що популярно серед ваших друзів, чим ви займалися в минулому і які Сторінки вам подобаються. Перегляд гайдів по різних місцях.
4. Можливість додавання календарів з телефону, щоб переглядати всі плани в одному місці.
5. Можливість підписатися на оновлення про найближчі заходи, щоб завжди бути в курсі всіх змін.

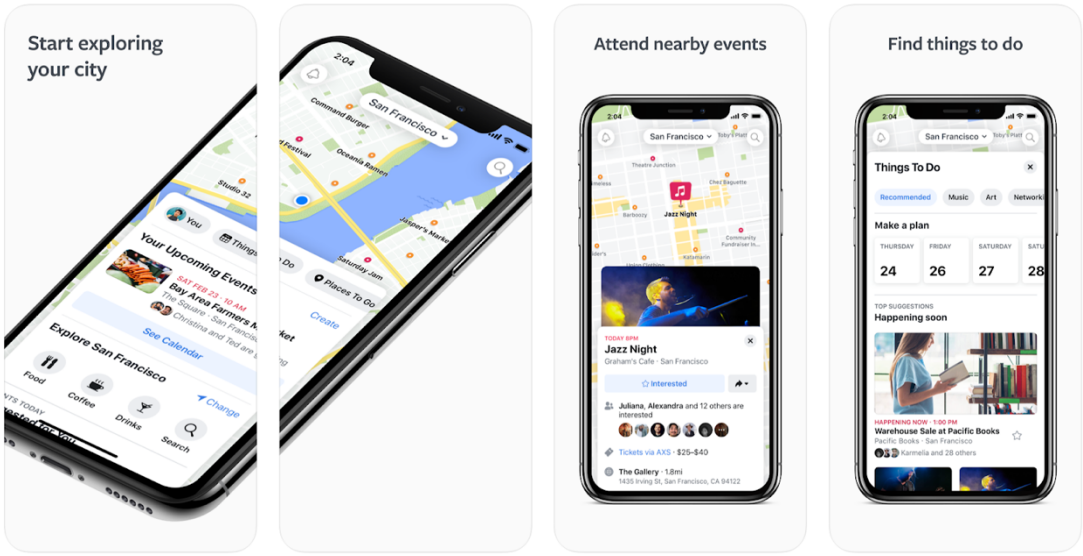


Рис. 1.1. Facebook local

1.4.2 MeetUp

Додаток Meetup - це платформа для пошуку та побудови локальних з'єднань. Ви можете використовувати цей додаток для зустрічей, щоб познайомитися з новими людьми, знайти підтримку, дізнатися нове, або вийти зі своєї зони комфорту. І ви можете відчути це в близьких місцях. Додаток використовує точне розташування пристрою (GPS та мережа), щоб рекомендувати зустрічі поблизу.

- Вибирайте цікаві категорії, шукайте за ключовими словами або дізнавайтеся, що популярно в вашому регіоні.

- Зберігайте цікаві заходи і стежте за ними.

- Залишайтеся на зв'язку з людьми, яких ви зустріли на заході, пишіть їм повідомлення і вступайте в обговорення.

- Перехід між різними містами і країнами, щоб подивитися, як там проводяться заходи.

Створіть групу.

- Організовуйте власні заходи, щоб знайомитися з людьми, яким цікаво те ж, що і вам.

- Призначайте заходи і керуйте групою звідки завгодно.

- Підтримуйте зв'язок з учасниками, викладаючи фотографії з заходів, задаючи їм питання і відкриваючи обговорення. Додаток допомагає створювати та приєднуватися до місцевих соціальних груп. Усі групи зосереджені навколо спільних інтересів. Наприклад, групи можуть бути для зустрічей та спілкування, тоді як інші групи можуть зайнятися кар'єрою, захопленнями та діловими мережами.

Додаток поєднаний з однією простою ідеєю: "коли ми збираємось разом і робимо важливі для нас речі, ми стаємо кращі". І це саме те, що додаток точно робить. Ось кілька популярних груп Meetup:

- Групи Tech Meetup
- Групи фітнес-зустрічі
- Походи груп зустрічі
- Кар'єра та мережа груп зустрічей
- Нові групи у місті Meetup
- Групи соціальних зустрічей та багато іншого.

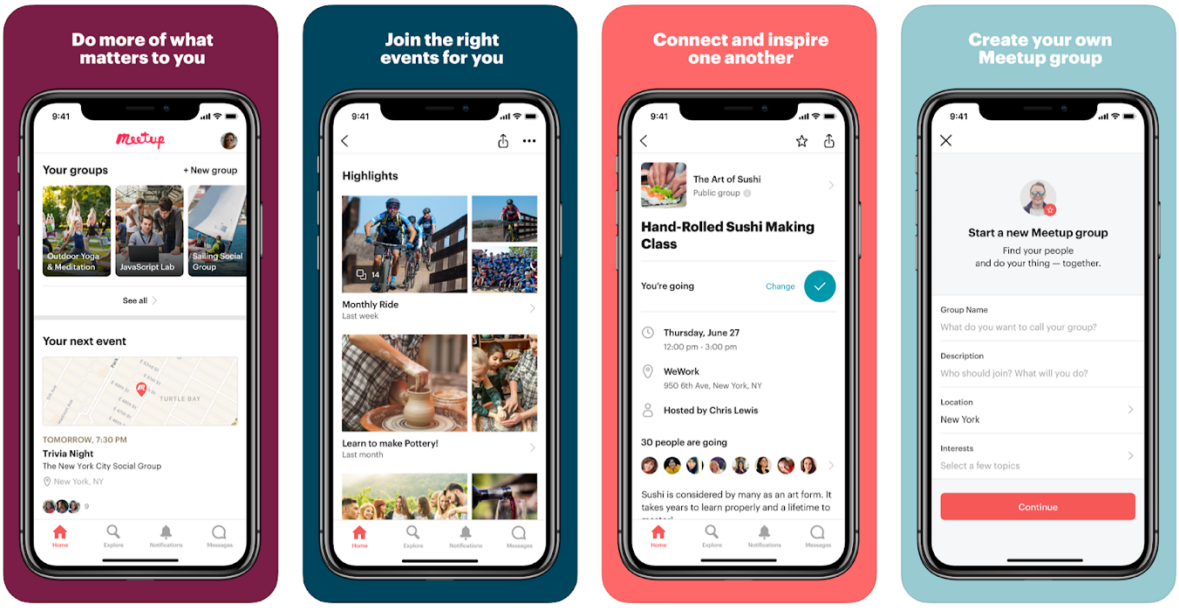


Рис. 1.2. MeetUp

1.4.3 NextDoor

Nextdoor - центр мікрорайону. Знайдіть поруч події, запропонуйте речі, які ви продаєте, або знайдіть чудові речі, доступні безкоштовно у вашому районі.

Використовуйте Nextdoor, щоб бути в курсі того, що відбувається у вашому районі. Зв'яжіться зі своїми сусідами, знайдіть домашні послуги, сформуйте годинник для сусідства або просто познайомтеся з людьми, яких ви бачите щодня.

Знайдіть розпродаж у дворі, майстра-професіонала, вигулу собак або няні. Спілкуйтеся зі своїми сусідами, щоб бути в курсі місцевих новин, планувати місцеві події або ділитися порадами з безпеки.

Від діяльності громади до продажу дворів, сусіди можуть нам так багато допомогти - нам просто потрібно з ними зв’язатися. Nextdoor дозволяє легко говорити з сусідами про те, що для вас найбільше важливо.

Місцевий додаток для сусідів

• Місцеві новини про всі події в околицях

• Отримайте рекомендації щодо кращого місцевого майстра

• Розміщуйте класифіковані оголошення, щоб продати сусідам

• Отримуйте пропозиції та безкоштовні речі, коли сусід пропонує цей використаний прилад, який їй більше не потрібен

• Зустріньтеся з сусідами і нарешті подзвоніть цього милого чоловіка по вулиці за своїм ім'ям

Місцеві події

• Знайдіть місцеві події, такі як кулінарія та громадські дії

• Знайдіть продажі гаражів та подвір’я біля себе

• Продаж нерухомості у вашому районі

Знайдіть домашні послуги та пропозиції

• Найміть місцеву няню або рекомендовану няню

• Найміть домашнього вихованця або прогулянків собак для свого пухнастого друга

• Знайдіть місцевого майстра професіонала, домашнього нагляду або громадського харчування в наших оголошеннях

Сповіщення про злочини та поради щодо безпеки

• Організуйте околиці, щоб швидко зрозуміти злочин та безпеку

• Ділитися інформацією під час стихійного лиха

• Звіт про злочини та карти, які допоможуть уберегти ваш мікрорайон

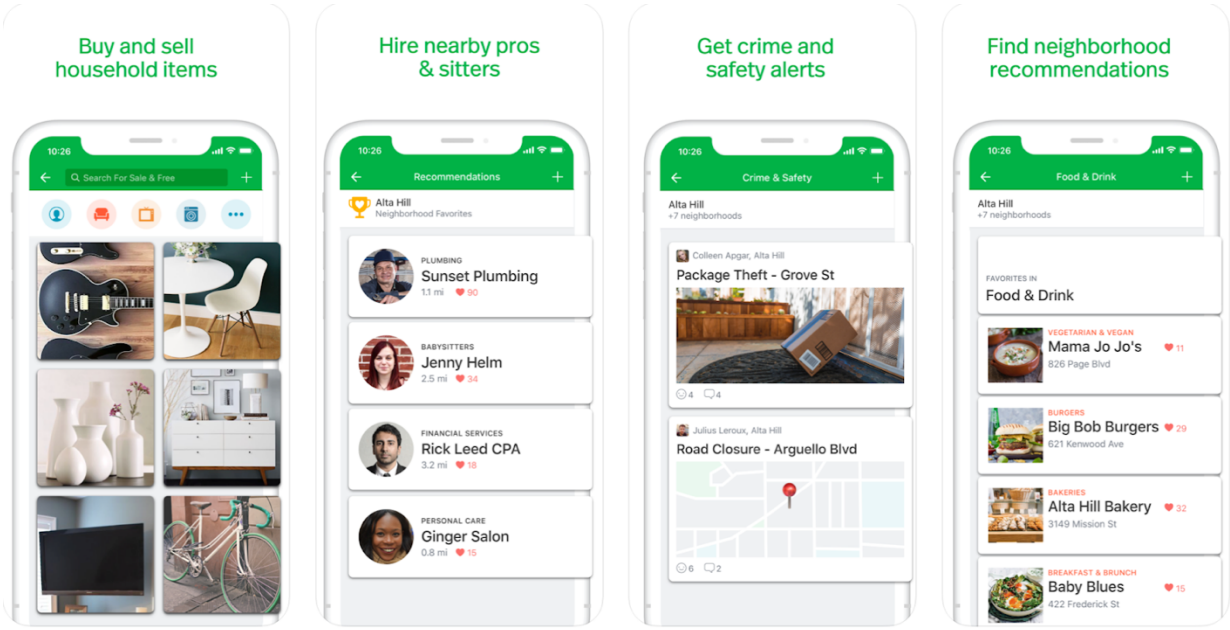


Рис. 1.3. NextDoor

1.4.4 Tinder

Це безкоштовний мобільний додаток для знайомств, який шукає вам пару у вашому районі. Tinder в основному запустив додаток для знайомств на основі місця розташування, і в 2012 році це найпопулярніший додаток для знайомств. Тільки з цієї причини, якщо ви самотні, Tinder, безумовно, варто використовувати.

Революційний (на той час) Tinder "проведіть пальцем праворуч, якщо вона вам подобається, проведіть пальцем ліворуч, якщо ні", з тих пір скопійовано численними конкурентами, і таких програм, як Tinder, є багато. Бамбл, CoffeeMeetsBagel - кілька прикладів.

Tinder в даний час є частиною групи Match, яка також володіє сайтами знайомств, як Plenty of Fish та Match.com [4].

Tinder - це програма для знайомств, яка дозволяє людям анонімно проводити пальцем ліворуч або праворуч на когось на основі фото, невеликої біографії та загальних інтересів. Провести пальцем ліворуч - це відхилити людину, а пальцем праворуч - виявити інтерес до відповідності. Якщо ви спілкуєтесь з людиною, ви можете обмінюватися повідомленнями та переглядати фотографії, які вони публікують в Інтернеті.

Оскільки "змах" є анонімним, люди не знають, чи ви проявили інтерес до них, поки ви не зрівняєтесь. Якщо двоє людей не відповідають, вони не можуть поговорити один з одним. Це запобігає можливості отримання повідомлень від людей, яких ви не цікавите.

Параметри Tinder

- Discovery: ця спеціальність повинна бути відкрита весь час для пошуку нових людей
- Місцезнаходження: це вказує поточне місцеположення, але ви можете провести пальцем куди завгодно, використовуючи преміальний рахунок
- Відстань: Вкажіть інтервал, з яким ви хочете зустрітися з людьми, він повинен бути менше 100 км для кращих результатів
- Віковий інтервал: виберіть вік, який шукаєте. Не забувайте, вас побачать люди, які визначили ваш вік для зустрічі

ВИСНОВКИ ДО РОДЗІЛУ 1

Сьогодні, напевно, ніхто не уявляє своє існування без інтернету. Зараз людині потрібно менше 15 хвилин, щоб знайти будь-яку інформацію. Раніше бібліотеки користувалися набагато більшим попитом. Люди збиралися разом та обговорювали спільні проблеми, знаходили спільні інтереси. Але з появою інтернету, люди почали все більше часу проводити там. Саме тому це сприяло появі перших соціальних мереж, де люди мають можливість спілкуватися не лише з однодумцями зі свого міста, а й з усього світу.

У світі безліч соціальних мереж. Перелік починається від FaceBook і закінчується соціальними мережами для придбання косметики. Все це стрімко віддаляє людей від реального світу та реального спілкування. Саме тому зараз стрімко розвиваються “антисоціальні мережі”, які, в свою чергу, є соціальними мережами, але які спонукають своїх користувачів зустрічатися та проводити час разом. Кожна “антисоціальна мережа” заточена під свою аудиторію.

MeetUp підходить тим людям, які шукають події, пов’язані з кар’єрним або професіональним ростом.

NextDoor створена для сусідів. Додаток допомагає дізнатися більше про сусідів у своїй місцевості та знайти спільну мову з ними.

Tinder допомагає знайти собі пару.

FaceBook Local створений для спільних розваг, подорожування.

РОЗДІЛ 2.
ВИБІР ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ ТА РОЗРОБКА

2.1 Вибір інструментів розробки програми

Swift – це мова програмування загального призначення, побудована з використанням нового підходу до моделей продуктивності, безпеки та програмного забезпечення.

Мета проекту Swift є створення найкращої доступної мови для застосувань, починаючи від системного програмування, закінчуючи настільними та мобільними програмами, масштабуючи до cloud сервісів. Найголовнішим є те, що Swift призначений для полегшення написання правильних програм для розробника. Для досягнення головної мети найбільш очевидним способом написання додатку на Swift також повинен бути:

Безпечним. Найбільш очевидний спосіб написання коду також повинен вести себе безпечно. Невизначена поведінка є ворогом безпеки, і помилки розробника повинні бути зафіксовані ще до того, як програмне забезпечення буде вироблятися [5].

Швидкий. Swift призначений як заміна для мов на основі C (C, C ++ та Objective-C). Таким чином, Swift повинен бути порівнянним з цими мовами у виконанні для більшості завдань. Продуктивність також повинна бути послідовною та передбачуваною.

Інструменти є важливою частиною екосистеми програмування на Свіфт. Apple прагне добре інтегруватися в набір інструментів розробника, швидко будувати, представляти найкращу діагностику та інтерактивний досвід розвитку.

Інструменти можуть зробити програмування набагато більш потужними, як playground на основі Swift в Xcode, або веб-базова REPL при роботі з кодом на Linux.

2.1.1 Особливості мови програмування Swift

Swift включає функції, які полегшують код для читання та запису, надаючи розробнику необхідний контроль у мові програмування системи.

Swift підтримує виведені типи, щоб зробити код чистішим і менш схильним до помилок, а модулі усувають заголовки та надають простори імен. Пам'ять керується автоматично. Swift також запозичує з інших мов, наприклад, названі параметри, висунуті з Objective-C, виражаються в чистому синтаксисі, що робить API в Swift легким для читання та обслуговування.

Особливості Swift розроблені для спільної роботи, щоб створити потужну, але приємну для використання мову. Деякі додаткові функції Swift включають:

- Замикання уніфіковані з функціональними покажчиками
- Кортежі та кілька повернених значень
- Дженріки
- Швидка і стисла ітерація в асортименті чи колекції
- Структури, які підтримують методи, розширення та протоколи
- Функціональні схеми програмування, наприклад, мапа та фільтр
- Потужна робота з вбудованою помилкою
- Розширений потік керування за допомогою ключових слів do, guard, defer та repeat

2.1.2 Безпечність мови програмування Swift

Ще одна особливість безпеки полягає в тому, що об'єкти Swift за замовчуванням ніколи не можуть бути нульовими, а спроба зробити або використовувати об’єкт nil призведе до помилки часу компіляції. Це робить код написання набагато більш чистим та безпечним та запобігає поширеній причині збоїв під час виконання. Однак є випадки, коли нуль підходить, і для цих ситуацій Swift має інноваційну функцію, відому як необов'язковий. Необов’язково може містити нуль, але синтаксис Swift змушує вас безпечно боротися з ним за допомогою ”?” щоб вказати компілятору, що ви розумієте поведінку і будете безпечно поводитися з нею [6].

2.2 Життевий цикл додатків IOS

- Не запущено

Додаток не був запущений або запущений, але він був повністю систематизований.

- Неактивний

Додаток працює на передньому плані, але він не отримує подій. (Можливо, він використовує інший код.) Додаток, який залишається у цьому стані, залишається ненадійним, коли він переходить до іншої стадії.

- Активний

Додаток працює на передньому плані та залишає події. Звичайний режим для додатків переднього плану.

- Задній план

Додаток знаходиться у фоновому режимі та виконує код. Більшість додатків переходять у цей стан на короткий час, коли вони зупиняються. Однак додаток, який вимагає додаткового часу виконання може залишатися в цьому стані протягом певного періоду часу. Крім того, додаток, що запускається безпосередньо у фоновому режимі - переходить у цей стан замість неактивного.

- Тимчасово припинений

Додаток знаходиться у фоновому режимі, але не виконує код. Система автоматично ставить додатки в цей стан і не повідомляє про них перед цим. Якщо програма призупинена, додаток залишається в пам'яті, але не виконує жодного коду. Якщо виникає стан низької пам’яті, система може очистити призупинені програми без попереднього повідомлення, щоб звільнити більше місця для додатка переднього плану.

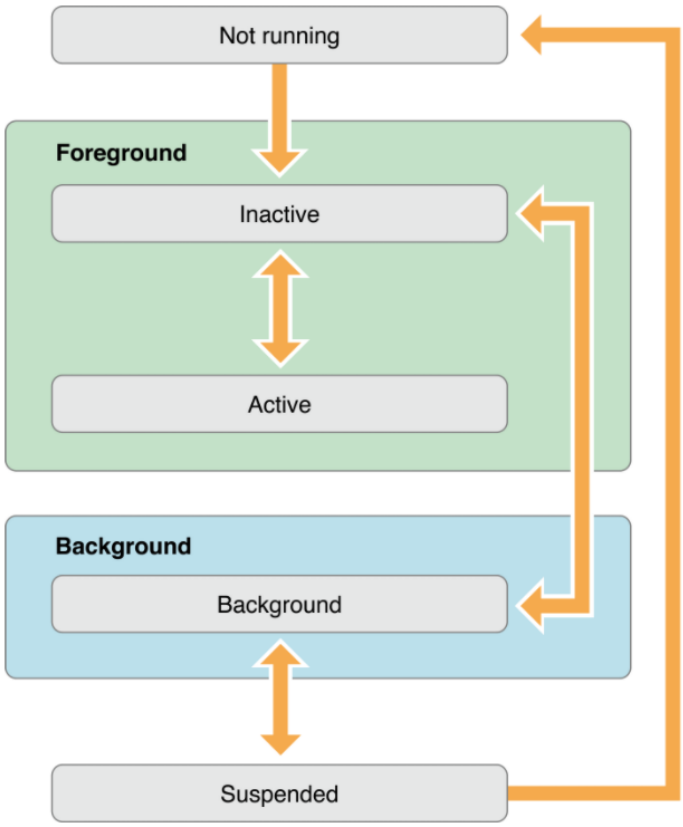


Рис. 2.1. Життєвий цикл додатку IOS

Кожен розробник iOS повинен знати життєвий цикл програми. Життєвий цикл програми допомагає зрозуміти загальну поведінку програми.

Основний пункт входу в додатки для iOS - це UIApplicationDelegate. UIApplicationDelegate - це протокол, який повинен застосувати ваш додаток, щоб отримувати сповіщення про події користувача, такі як : запуск додатка, додаток переходить у другий план або передній план, додаток припиняється, відкрите push-повідомлення тощо.

Коли запускається додаток iOS, перше, що визивається – це application: willFinishLaunchingWithOptions:-> Bool.

Цей метод призначений для початкової настройки програми. Storyboards уже завантажені, але відновлення стану ще не відбулося.

Старт:

- **application: didFinishLaunchingWithOptions: -> Bool**

Цей метод зворотного виклику викликається, коли програма закінчила запуск і відновила стан і може зробити остаточну ініціалізацію, таку як створення інтерфейсу користувача.

- **applicationWillEnterForeground**: викликається після: **didFinishLaunchingWithOptions**: або якщо ваш додаток знову активується після отримання телефонного дзвінка чи іншої системної перерви.
- **applicationDidBecomeActive**: викликається після **applicationWillEnterForeground**: для завершення переходу на передній план.

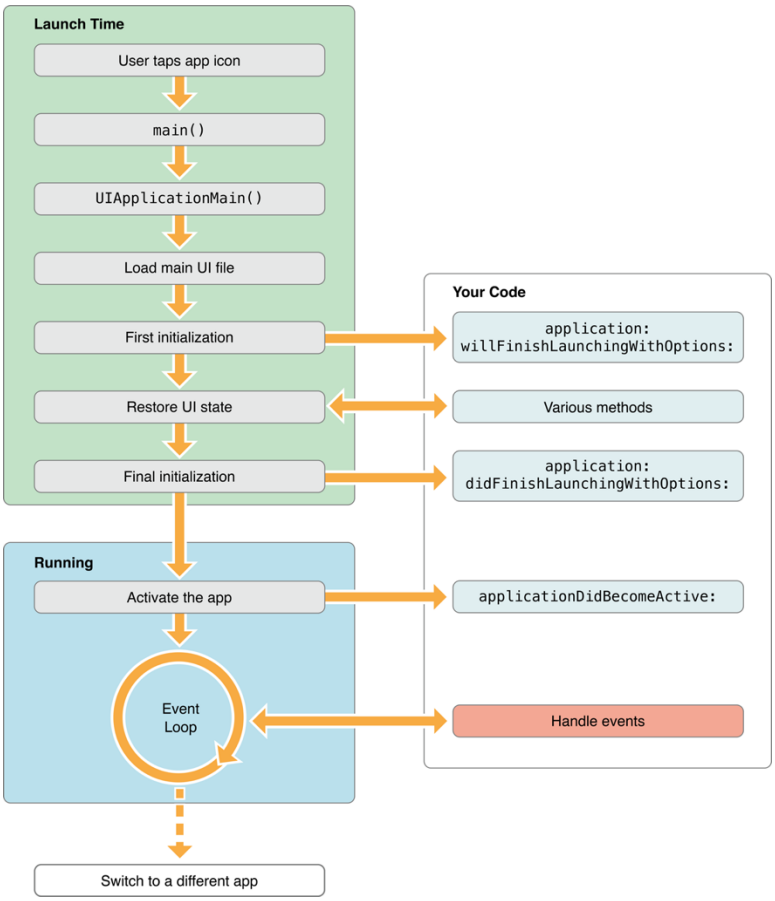


Рис. 2.2. Основний режим додатку IOS

Припинення:

- **applicationWillResignActive**: викликається, коли програма збирається стати неактивною (наприклад, коли телефон отримує дзвінок або користувач натискає кнопку «Головна»).
- **applicationDidEnterBackground**: викликається, коли додаток переходить у фоновий стан після неактивності. У вас є приблизно п’ять секунд для виконання будь-яких завдань, необхідних для резервного копіювання речей у випадку, якщо додаток припиняється пізніше або відразу після цього.
- **applicationWillTerminate**: викликається, коли ваш додаток буде очищено з пам'яті. Викликайте на будь-які остаточні очищення тут.

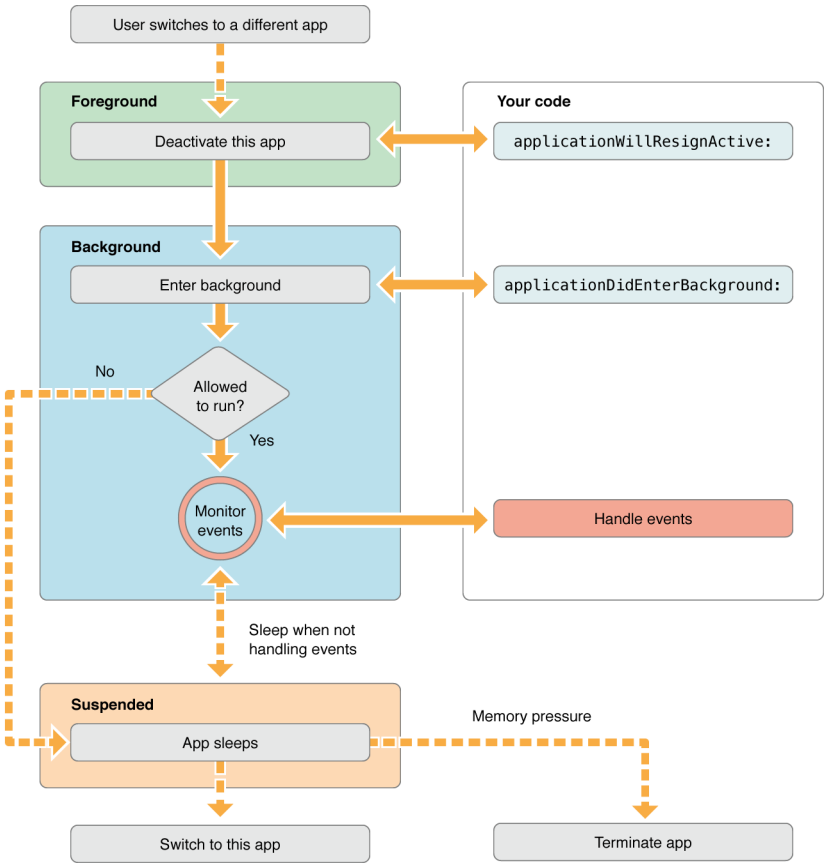


Рис. 2.2. Завершення циклу додатку IOS

2.3 Життевий цикл UIViewController

Методи життєвого циклу:

. loadView()

. loadViewIfNeeded()

. viewDidLoad()

. viewWillAppear(_ animated: Bool)

. viewWillLayoutSubviews()

. viewDidLayoutSubviews()

. viewDidAppear(_ animated: Bool)

. viewWillDisappear(_ animated: Bool)

. viewDidDisappear(_ animated: Bool)

- loadView ()

Ця подія створює подання, яким керує контролер. Він викликається лише тоді, коли контролер перегляду створений програмно [7].

Можна перевизначити цей метод, щоб створити свої представлення вручну.

- loadViewIfNeeded ()

Завантажує переглчяд, якщо воно ще не встановлено.

доступний з iOS> = 9.0

- viewDidLoad ()

Викликається після завантаження перегляду.

Для контролерів перегляду, створених у коді, це після -loadView.

Використовують цей метод для ініціалізації налаштування інтерфейсу

- viewWillAppear (_ анімований: Bool)

Цей метод визиватиметься кожного разу, коли огляд переглянеться, незалежно від того, чи є перегляд вже в пам'яті.

- viewWillLayoutSubviews ()

Викликається безпосередньо перед layoutSubviews

Це перший крок у життєвому циклі, де встановлюються межі (bounds).

- viewDidLayoutSubviews ()

Викликається одразу після layoutSubviews

Ця подія сповіщає контролер перегляду, що внутрішні перегляди були налаштовані.

- viewDidAppear (_ анімований: Bool)

Викликається, коли погляд повністю переведений на екран.

- viewWillDisappear (_ анімований: Bool)

Цей метод буде викликаний, коли перегляд контролера перегляду буде видалено з ієрархії подання.

- viewDidDisappear (_ анімований: Bool)

Цей метод буде викликаний, коли вигляд контролера перегляду видалений з ієрархії подання.

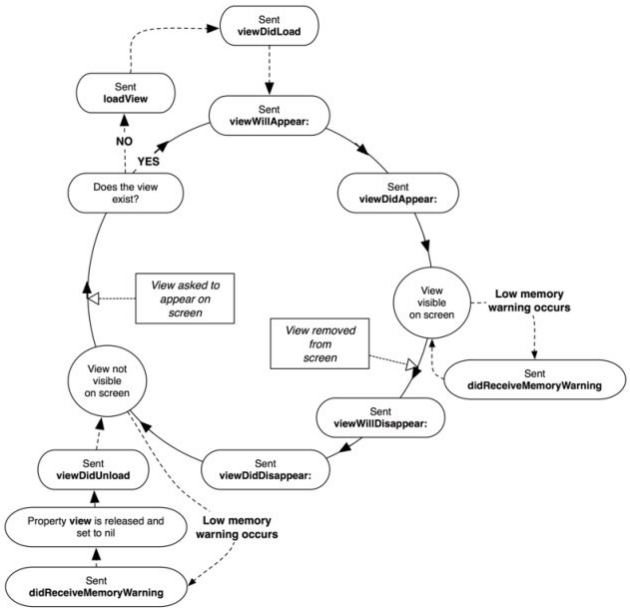


Рис. 2.3. Цикл UIViewController

2.4 Ієрархія переглядів

Програми часто містять багато представлень і більше одного контролера перегляду, створюючи дві ієрархії:

- Ієрархія UIView - це дерево з коренем у Window (UIWindow - підклас UIView). Ми можемо обходити це дерево, використовуючи властивості subviews та superview.
- Ієрархія UIViewController починається від rootViewController Window. Контролер перегляду може представленим або є контейнером для інших контролерів перегляду, створюючи дві різні ієрархії [8].

2.4.1 Представлені контролери перегляду

Коли ми відображаємо контролер перегляду, використовуючи функцію Present Modally або Preset As Popover Segue або present, ми створюємо презентаційне → представлене з'єднання.

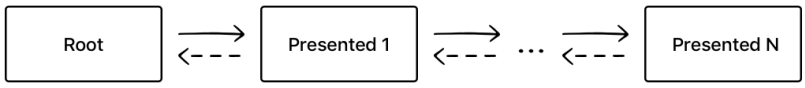


Рис. 2.4. Представлені контролери

Представлені контролери перегляду складають такий собі стек: останній контролер, який ми бачимо - зазвичай відхиляється першим.

Представлені контролери подання з'єднуються у подвійному зв’язку зі списком presentedViewController та властивостями presentingViewController .

2.4.2 Контейнер контролерів перегляду

Контейнер управляє іншими контролерами перегляду, якими він володіє. У Storyboard ми використовуємо метод Show та Show Detail або можемо використовувати метод addChild (_ :), створюючи батьківське → дочірнє з'єднання.

Це утворює дерево, яке можна перетинати з властивостями дітей та батьків.

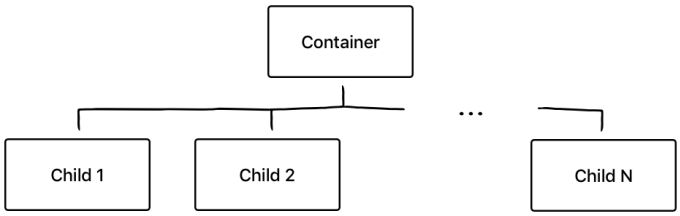


Рис. 2.4. Контейнер переглядів

Багато розповсюджених контролерів перегляду є контейнерами: UINavigationController, UITabBarController, UISplitViewController, UIPageViewController.

Комбінація можлива, тобто коли контейнер представляє інший контролер перегляду. Але неможливо, щоб контролер перегляду був представлений і в контейнері одночасно.

Коли контейнер представлений, ми отримуємо комбінацію двох структур, представлений контролер перегляду може утворювати інше дерево.

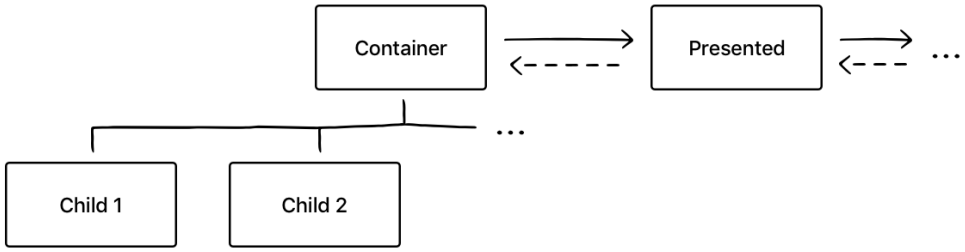


Рис. 2.5. Комбінація переглядів

Відмінності між двома ієрархіями стають чіткими, коли мова йде про обробку подій. У дереві діти рівноцінні, а у стеку пріоритет має верх.

2.5 Core Location

Core Location надає можливості, які визначають географічне розташування пристрою, висоту та орієнтацію пристрою або його положення відносно пристрою iBeacon, що знаходиться поблизу. Core збирає дані, використовуючи всі доступні компоненти на пристрої, включаючи Wi-Fi, GPS, Bluetooth, магнітометр, барометр.

Використовується екземпляри класу CLLocationManager для налаштування, запуску та зупинки служб Core Location. Об'єкт диспетчера локацій підтримує такі дії, пов'язані з розташуванням:

Стандартні та значні оновлення місцеположення. Відстежуйте великі або невеликі зміни в поточному місці користувача з можливістю налаштування точності.

Моніторинг регіону. Відстежуйте окремі регіони, що цікавлять, і генеруйте події місцеположення, коли користувач входить або залишає ці регіони.

“Маяк” діапазону. Виявити та знайти “маяки” поблизу.

Заголовки компасів. Зміни заголовка звітів із бортового компаса.

На пристроях iOS користувачі можуть в будь-який час змінювати налаштування служби локації в додатку Налаштування, впливаючи на окремі додатки або пристрій у цілому. Ваш додаток отримує події, включаючи зміни авторизації, в об’єкт делегата вашого менеджера місцеположення, який відповідає протоколу CLLocationManagerDelegate.

Apple вважає місцезнаходження користувача приватним, і це означає, що нам потрібно попросити дозволу на його використання. Як ви просите дозволу, залежить від того, що ви намагаєтеся зробити: чи хотілося б вам розташування користувача лише тоді, коли ваша програма запущена, чи ви хочете розташування користувача, навіть коли ваш додаток не працює?

Існує п’ять рівнів доступу для перевірки [9].

- дозволено завжди
- дозволено додаток відкритий
- заборонено
- не визначено
- обмежений

Якщо ви створюєте додаток для карти, який показує користувачам, як дістатися з поточного місця розташування до найближчого магазину, вам

знадобиться їх місцезнаходження лише тоді, коли додаток використовується. Але якщо ви створюєте додаток, який потрібно розбудити, коли користувач досягне місця, вам знадобиться доступ навіть тоді, коли програма не працює - iOS відстежує місцезнаходження користувача від вашого імені та автоматично запускає ваш додаток як потрібно.

Використання місця, коли додаток не запущено, звичайно, є дуже чутливою інформацією, тому Apple позначає її трьома способами:

Якщо ви запитаєте “Завжди доступ”, користувачі все одно отримають можливість вибору "Коли використовується".

Якщо вони обирають “Завжди”, iOS автоматично запитає їх знову через кілька днів, щоб підтвердити, що вони все ще хочуть надати “Завжди доступ”.

Коли ваш додаток використовує дані про місцезнаходження у фоновому режимі, інтерфейс iOS оновиться.

Користувачі можуть у будь-який момент зайти в додаток налаштувань і змінити з "Завжди" на "Коли використовується".

2.6 MapKit

MapKit існує для відображення карти чи супутникових знімків з інтерфейсу програми, називайте цікаві місця та визначайте інформацію про орієнтири для координат карти [10].

MapKit використовується, щоб вставляти карти безпосередньо у власні вікна та представлення. Можна додавати на карту “примітки” та “накладки”, щоб запам’ятати визначні місця або пункти призначення користувачів

Якщо додаток пропонує маршрутні маршрути, можна зробити свої вказівки доступними для Карт. Також можна використовувати Карти, щоб доповнити вказівки, які ви надаєте у своєму додатку. Наприклад, якщо додаток надає лише вказівки для подорожі в метро, можна використовувати Карти, щоб вказати маршрути до станцій метро і від них.

2.7 URL сесії

Клас URLSession та пов’язані з ним класи надають API для завантаження даних із та завантаження даних у кінцеві точки, вказані URL-адресами. Додаток також може використовувати цей API для виконання фонових завантажень, коли додаток не працює або призупинений. Можна використовувати URLSessionDelegate та URLSessionTaskDelegate для підтримки автентифікації та отримання таких подій, як перенаправлення та task completion.

Додаток створює один або кілька екземплярів URLSession, кожен з яких координує групу пов’язаних із цим завдань щодо передачі даних. Наприклад, якщо ви створюєте веб-браузер, додаток може створити один сеанс на вкладці чи вікні або один сеанс для інтерактивного використання та інший для фонового завантаження. Протягом кожного сеансу додаток додає низку завдань, кожне з яких представляє запит на певну URL-адресу (якщо потрібно, перенаправлення HTTP) [11].

2.7.1 Типи URL сесій

Завдання в межах певного сеансу URL-адреси мають спільний об'єкт конфігурації сеансу, який визначає поведінку з'єднання, максимальну кількість одночасних з'єднань для одного хоста, чи можуть з'єднання використовувати спільну мережу тощо.

У URLSession є єдиний спільний сеанс (у якого немає об’єкта конфігурації) для основних запитів. Він не дуже розширюється, але може слугувати гарною відправною точкою, якщо у вас дуже обмежені вимоги. Ви можете отримати доступ до цього сеансу, викликавши метод спільного класу. Для інших типів сеансів ви створюєте URL-сесію з одним із трьох типів конфігурацій:

- Сеанс за замовчуванням веде себе як спільний сеанс, але дозволяє налаштувати його. Ви також можете призначити делегата сеансу за замовчуванням, щоб поступово отримувати дані.
- Сеанси, які не записують файли cookie чи персональні дані на диск.

- Фонові сеанси дозволяють виконувати завантаження та завантаження вмісту у фоновому режимі, поки ваша програма не працює.

2.7.2 Типи тасків URL сесії

У межах сеансу ви створюєте таски, які опціонально завантажують дані на сервер, а потім витягують дані з сервера або як файл на диску, або як один або кілька об'єктів NSData в пам'яті. API URLSession пропонує чотири типи тасків:

- Data tasks надсилають та отримують дані за допомогою об'єктів NSData.
- Вони призначені для коротких, часто інтерактивних запитів до сервера.
- Upload tasks схожі на завдання з передачею даних, але вони також надсилають дані (часто у вигляді файлу) та підтримують фонове завантаження, коли програма не працює.
- Download tasks, отримуйте дані у вигляді файлу, а також підтримуйте завантаження та завантаження фонових зображень, поки програма не працює.
- WebSocket tasks обмінюються повідомленнями через TCP та TLS, використовуючи протокол WebSocket, визначений в RFC 6455.

2.7.3 Використання делегату URL сесії

Таски сеансу також розділюють спільний об’єкт делегата [12]. Ви реалізуєте цього делегата для надання та отримання інформації, коли відбуваються різні події, у тому числі, коли:

- Аутентифікація не вдається.
- Дані надходять із сервера.
- Дані стають доступними для кешування.

Об'єкт сеансу зберігає strong посилання на делегата, поки ваша програма не буде закритою або явно не визнає недійсним сеанс. Якщо ви не визнаєте недійсним сеанс, ваш додаток буде неефективно розпоряджатися пам’ятю допоки він не припнеться.

2.7.4 Асинхронність

Як і більшість мережевих API, API URLSession є асинхронним. Він повертає дані у ваш додаток одним із двох способів, залежно від методів, які ви визиваєте:

- Викликаючи блок обробника завершення, коли передача закінчується успішно або з помилкою.
- Викликаючи методи делегата сеансу, коли дані надходять і коли передача буде завершена.

URLSession надає властивості статусу та прогресу, які ви можете запитувати, якщо вам потрібно приймати програмні рішення, виходячи з поточного стану таска (з застереженням, що його стан може змінитися в будь-який час).

2.7.4 Потокобезпечність

API сеансу URL потоко-захищений. Ви можете вільно створювати сесії та таски в будь-якому потоку [13]. Коли ваші методи делегата викликають надані обробники завершення, робота автоматично планується в правильній черзі делегата.

ВИСНОВКИ ДО РОЗДІЛУ 2

Раніше будь-який IOS додаток розроблявся на мові Objective-C. Розробка була незручною та повільною. З появою Swift все змінилося. Дані показують, що Swift є значно більш коротким, ніж Objective-C, і пропонує розробникам спосіб заміни близько 70% коду для досягнення тих же результатів. З точки зору кодування та розробки - це величезна перевага. Розглядаючи загальну швидкість та продуктивність, Swift на 2,6 рази швидший, ніж Objective-C. Swift пропонує динамічні бібліотеки, що є великою перевагою. Динамічні бібліотеки кращі, ніж статичні, з кількох різних причин. Статичні бібліотеки фіксуються в коді при компіляції програми. Це додає розміру програми та збільшує час її завантаження. Ще один мінус для статичних бібліотек полягає в тому, що вони не оновлюються автоматично.

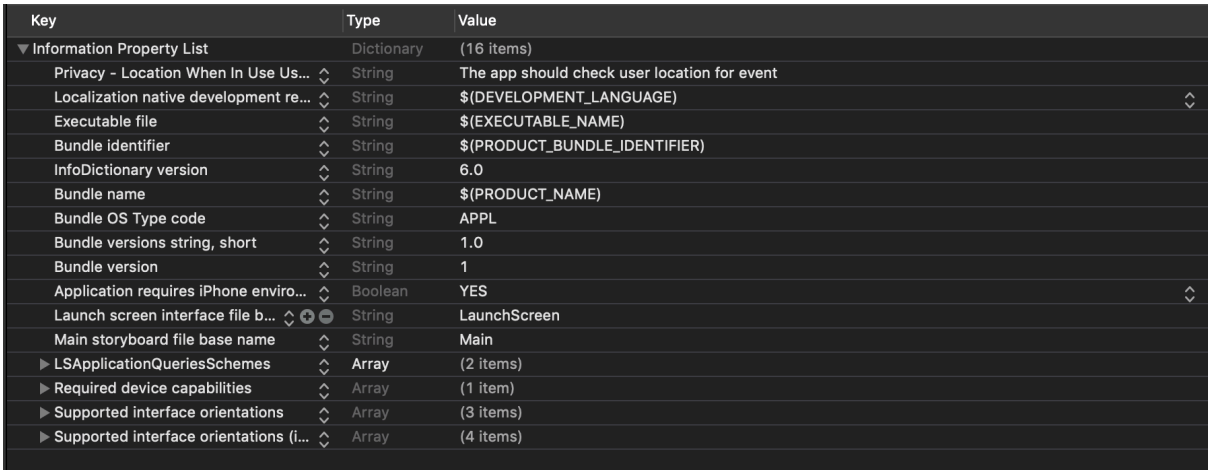
І навпаки, динамічні бібліотеки, які існують поза кодом, завантажуються лише за потреби. Саме це допомагає зменшити загальний розмір програми. Swift пропонує простий і простий спосіб управління пам'яттю додатків. Ця функція допомагає запобігти проблем з пам'яттю за допомогою автоматичного підрахунку посилань. За допомогою автоматичного підрахунку пам'яті (ARC) Swift визначає, які екземпляри більше не потрібні. ARC автоматично позбавиться від них, допомагає покращити продуктивність додатків, не впливаючи на пам'ять або процесор.

РОЗДІЛ 3.
РОЗРОБКА СИСТЕМИ

Розробка будь-якого додатка на платформу IOS повинна починатися з файлу info.plist.

Список властивостей або plist - це XML-файл, який містить дані ключових значень. Найпростіше порівняти зі словником у Swift, так що це список значень, пов'язаних з ключами.

Список властивостей може містити кілька типів елементів, включаючи масиви, словники, булеві значення, Data, числа. Кожен з цих типів відповідає рідним типам Swift, таким як Array та Bool. Ви також можете вкладати масиви та словники, тобто додавати масив всередині іншого масиву тощо, щоб створити більш складні структури даних.



Key	Type	Value
Information Property List	Dictionary	(16 items)
Privacy - Location When In Use Us...	String	The app should check user location for event
Localization native development re...	String	$(DEVELOPMENT_LANGUAGE)
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
Launch screen interface file b...	String	LaunchScreen
Main storyboard file base name	String	Main
LSApplicationQueriesSchemes	Array	(2 items)
Required device capabilities	Array	(1 item)
Supported interface orientations	Array	(3 items)
Supported interface orientations (i...	Array	(4 items)

Рис. 3.1. Plist Файл

У додатку "Events App" зконфігуровано " Launch screen interface file base name " – LaunchScreen. Це ім'я файлу, до я кого ми звернемося, при початковому запуску додатку.

Середній час, який займає початковий запуск є 2 секунди, але користувач не повинен застрягати на "чорному екрані", тому в окремому потоці запускається LaunchScreen, в якому можуть бути навіть анімації.

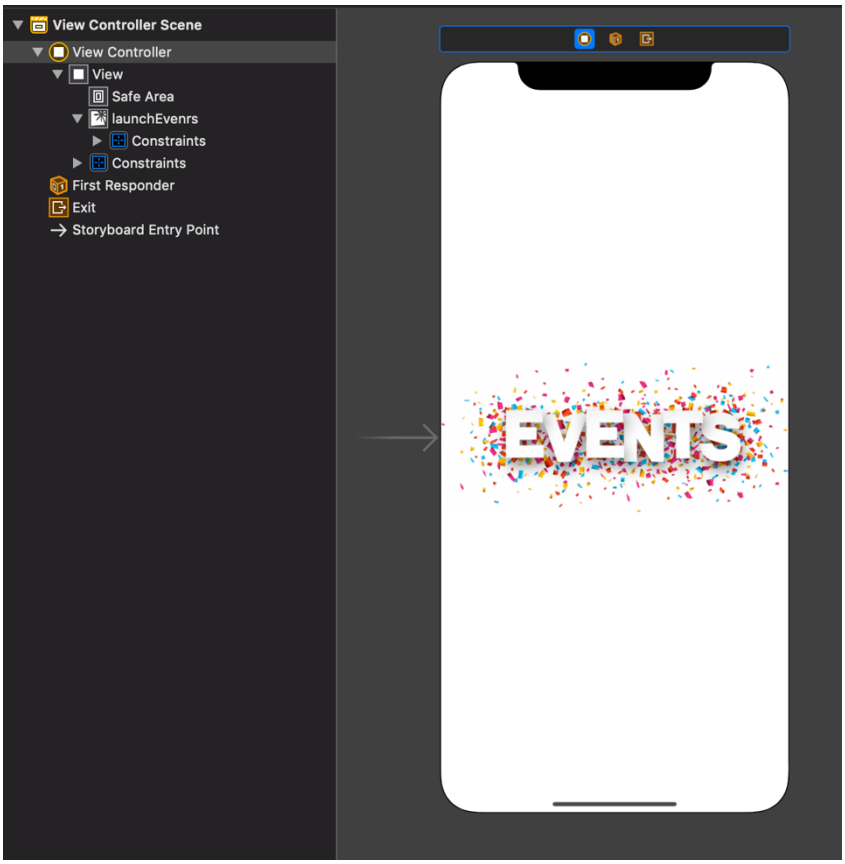


Рис. 3.2. Launch Екран

Саме тут починається життєвий цикл додатку.

Стартує :

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: **Any**]?) -> Bool

Тут ми підгружаємо з пам’яті всі потрібні данні та розпоряджаємося як додатку потрібно слідувати.

Додано “Main storyboard file base name” – Main. Це ім’я файлу, до я кого ми звернемося, після того, як додаток буде готовий до використовування (після LaunchScreen).

Також додана властивість “Privacy - Location When In Use Usage Description”, яка дозволить використовувати LocationManager.

3.1 Взаємодія з BackEnd

3.1.1 Структура

Створюючи що-небудь, завжди важливо мати структуру. Структура папок є ключовим фактором архітектури програмного забезпечення. Файли повинні були добре організованими.

Огляд структури взаємодії з BackEnd:

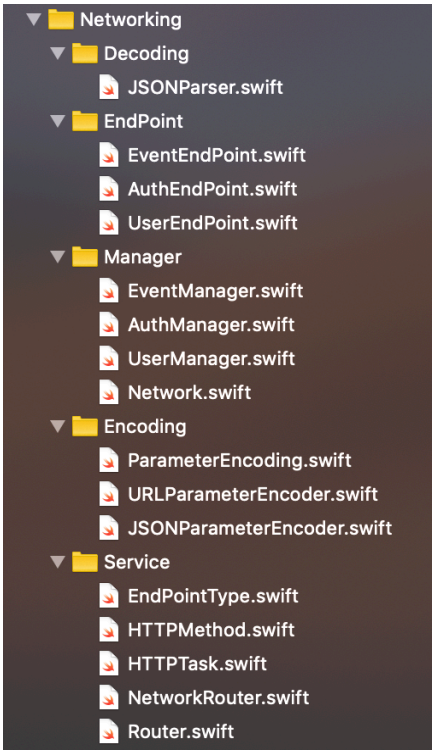


Рис. 3.3. Структура

3.1.1 EndPointType Протокол

Перше, що нам потрібно - це визначити наш протокол EndPointType. Цей протокол буде містити всю інформацію для налаштування EndPoint. Що таке EndPoint? Це URLRequest з усіма його компонентами, такими як заголовки, параметри запиту та параметри тіла. Протокол EndPointType є коренем нашої реалізації взаємодії з BackEnd.

```swift
protocol EndPointType {
    var baseURL: URL { get }
    var path: String { get }
    var httpMethod: HTTPMethod { get }
    var task: HTTPTask { get }
    var headers: HTTPHeaders? { get }
}
```

Рис. 3.4. Реалізація EndPointType

3.2 HTTP Протокол

Наш EndPointType має ряд протоколів HTTP, які нам потрібні для створення взаємодії з BackEnd.

3.2.1 HTTP Метод

- GET використовується для запиту даних із визначеного ресурсу.
- POST використовується для надсилання даних на сервер для створення / оновлення ресурсу.
- PUT використовується для надсилання даних на сервер для створення / оновлення ресурсу.
- Метод PATCH використовується для часткової зміни існуючого ресурсу.
- Метод DELETE видаляє вказаний ресурс.

```swift
public enum HTTPMethod : String {
    case get     = "GET"
    case post    = "POST"
    case put     = "PUT"
    case patch   = "PATCH"
    case delete  = "DELETE"
}
```

Рис. 3.5 Реалізація HTTP метод

3.2.2 HTTP Task

HTTPTask відповідає за налаштування параметрів для EndPoint. Можна додавати різні типи реквестів під різні задачі. Для моєї задачі потрібно лише 3 типів реквестів:

- Звичайний реквест

  Реквест з параметрами

- Реквкст з параметрами та заголовками

HTTP заголовки мають структуру [String : String], тому для спрощення коду створимо тип HTTPHeaders.

```
public typealias HTTPHeaders = [String:String]

public enum HTTPTask {
    case request

    case requestParameters(bodyParameters: Parameters?,
        bodyEncoding: ParameterEncoding,
        urlParameters: Parameters?)

    case requestParametersAndHeaders(bodyParameters: Parameters?,
        bodyEncoding: ParameterEncoding,
        urlParameters: Parameters?,
        additionHeaders: HTTPHeaders?)
}
```

Рис. 3.6 Реалізація HTTPTask

3.3 Параметри та кодування

Структура параметрів має бути [String : Any], так як ключ повинен бути строкою, а значення будь-яким.

Далі визначаю протокол ParameterEncoder з однією статичною функцією encode . Метод кодування приймає два параметри : URLRequest та параметри.

INOUT - це ключове слово Swift, яке визначає аргумент як аргумент-посилання. Зазвичай змінні передаються функціям як типи значень [14]. Розміщуючи inout перед аргументом, ми визначаємо його як тип посилання. Протокол ParameterEncoder буде реалізований JSONParameterEncoder та URLPameterEncoder.

```swift
public typealias Parameters = [String:Any]

public protocol ParameterEncoder {
    func encode(urlRequest: inout URLRequest, with parameters: Parameters) throws
}
```

Рис. 3.7 Реалізація ParameterEncoder

ParameterEncoder виконує одну функцію, яка кодуваннє параметри. Цей метод може закінчитися успішно, тому він може видати помилку, і потрібно це передбачити.

Для цього я створюю enum, який успадковується від Error.

```swift
public enum NetworkError : String, Error {
    case parametersNil = "Parameters were nil."
    case encodingFailed = "Parameter encoding failed."
    case missingURL = "URL is nil."
}
```

Рис. 3.8 Реалізація NetworkError

3.3.1 Кодування параметрів URL

Створено структуру “URLParameterEncoder”, який імплементує “ParameterEncoder”.

Спочатку ми повинні переконатися, що url існує, інакше оповістити користувача про помилку, використовуючи enum “NetworkError”. Далі кожен параметр, який будо передано до функції додаємо в url компоненти та кодуємо. І нарешті додаємо готові компоненти до посилання на URL реквест.

Програма приймає параметри і забезпечує їх безпечне передавання як параметри URL-адреси. Як слід знати, деякі символи заборонені в URL-адресах. Параметри також розділені символом '&', тому потрібно впоратись із усім цим. Для запиту також додано відповідні заголовки, якщо вони не встановлені.

```swift
public struct URLParameterEncoder: ParameterEncoder {
    public func encode(urlRequest: inout URLRequest, with parameters: Parameters) throws {

        guard let url = urlRequest.url else { throw NetworkError.missingURL }

        if var urlComponents = URLComponents(url: url,
                                             resolvingAgainstBaseURL: false), !parameters.isEmpty {

            urlComponents.queryItems = [URLQueryItem]()

            for (key,value) in parameters {
                let queryItem = URLQueryItem(name: key,
                                             value: "\(value)".addingPercentEncoding(withAllowedCharacters:
                                                 .urlHostAllowed))
                urlComponents.queryItems?.append(queryItem)
            }
            urlRequest.url = urlComponents.url
        }

        if urlRequest.value(forHTTPHeaderField: "Content-Type") == nil {
            urlRequest.setValue("application/x-www-form-urlencoded; charset=utf-8", forHTTPHeaderField:
                "Content-Type")
        }

    }
}
```

Рис. 3.9 Реалізація URLParameterEncoder

3.3.2 Кодування параметрів JSON

Програма повинна підтримувати різні види кодування параметрів, тому було створено структуру “JSONParameterEncoder”, яка підтримує кодування JSON параметрів, якщо цього вимагає BackEnd.

```swift
public struct JSONParameterEncoder: ParameterEncoder {
    public func encode(urlRequest: inout URLRequest, with parameters: Parameters) throws {
        do {
            let jsonAsData = try JSONSerialization.data(withJSONObject: parameters, options: .prettyPrinted)
            urlRequest.httpBody = jsonAsData
            if urlRequest.value(forHTTPHeaderField: "Content-Type") == nil {
                urlRequest.setValue("application/json", forHTTPHeaderField: "Content-Type")
            }
        }catch {
            throw NetworkError.encodingFailed
        }
    }
}
```

Рис. 3.10 Реалізація JSONParameterEncoder

Тепер створимо enum, у якого буде 3 типи параметрів:

- URL кодування
- JSON кодування
- URL та JSON кодування

Та у функції encode створимо switch, який буде кодувати параметри, знаючи, який саме метод потрібно вибрати. Якщо є body параметри, то

кодувати за допомогою JSON кодування. Якщо є url параметри, то кодувати за допомогою URL кодування.

А якщо потрібно в запиті і url і body параметри, то кодуємо за допомогою обох кодувань. Ця функція відповідає за кодування наших параметрів. Оскільки наш API очікує, що всі bodyParameters як JSON та URLParameters як URL-адреса, ми просто передаємо відповідні параметри його призначеному кодеру.

```
public enum ParameterEncoding {

    case urlEncoding
    case jsonEncoding
    case urlAndJsonEncoding

    public func encode(urlRequest: inout URLRequest,
                       bodyParameters: Parameters?,
                       urlParameters: Parameters?) throws {
        do {
            switch self {
            case .urlEncoding:
                guard let urlParameters = urlParameters else { return }
                try URLParameterEncoder().encode(urlRequest: &urlRequest, with: urlParameters)

            case .jsonEncoding:
                guard let bodyParameters = bodyParameters else { return }
                try JSONParameterEncoder().encode(urlRequest: &urlRequest, with: bodyParameters)

            case .urlAndJsonEncoding:
                guard let bodyParameters = bodyParameters,
                    let urlParameters = urlParameters else { return }
                try URLParameterEncoder().encode(urlRequest: &urlRequest, with: urlParameters)
                try JSONParameterEncoder().encode(urlRequest: &urlRequest, with: bodyParameters)

            }
        }catch {
            throw error
        }
    }
}
```

Рис. 3.11 Реалізація ParameterEncoding

3.4 Роутер взаємодії з BackEnd

NetworkRouter має EndPoint, який використовує для подання запитів, і коли запит зроблений, він передає відповідь на завершення. Відповідь має мітити данні у форматі Data та тип відповіді, але додано ще тип Error, якщо трапилась помилка. Також додана функція запиту. Цю функцію можна викликати в будь-який час у життєвому циклі запиту та скасувати її. Це може виявитися дуже цінним, якщо у програмі є завдання з завантаженням файлів. Тут ми використовуємо асоційований тип, оскільки хочемо, щоб наш маршрутизатор міг обробляти будь-який EndPointType. Без використання асоційованого типу роутер повинен мати конкретний EndPointType.

```swift
public typealias NetworkRouterCompletion = (_ data: Data?,_ response: URLResponse?,_ error: Error?)->()

protocol NetworkRouter: class {
    associatedtype EndPoint: EndPointType
    func request(_ route: EndPoint, completion: @escaping NetworkRouterCompletion)
    func cancel()
}
```

Рис. 3.12 Реалізація NetworkRouter

3.4.1 Реквест

Тут я створюю URL-сесію за допомогою shared сесії. Це найпростіший спосіб створення URL-сесії. Але більш складні конфігурації URL-сесії можуть бути реалізовані за допомогою конфігурацій, які можуть змінювати поведінку сеансу.

Тут ми створюємо наш запит, викликаючи buildRequest і надаючи йому маршрут, який є EndPoint. Цей виклик сховаємо в блок do-try-catch як buildRequest, оскільки помилка може бути видана нашими кодерами. Ми просто передаємо всі відповіді, дані та помилки до завершення.

```swift
class Router<EndPoint: EndPointType>: NetworkRouter {
    private var task: URLSessionTask?

    func request(_ route: EndPoint, completion: @escaping NetworkRouterCompletion) {

        let session = URLSession.shared
        do {
            let request = try self.buildRequest(from: route)
            //NetworkLogger.log(request: request)
            task = session.dataTask(with: request, completionHandler: { data, response, error in
                completion(data, response, error)
            })
        }catch {
            completion(nil, nil, error)
        }
        self.task?.resume()
    }

    func cancel() {
        self.task?.cancel()
    }
```

Рис. 3.13 Реалізація Роутера

3.4.2 Будуємо Реквест

Створюю приватну функцію всередині маршрутизатора під назвою buildRequest. Ця функція відповідає за всю життєво важливу роботу нашого мережевого рівня. По суті перетворення EndPointType в URLRequest. Як тільки наша EndPoint стає запитом, ми можемо передати її до сеансу.

Розглянемо кожен метод окремо. Розберемо метод buildRequest:

1. Створюємо запит змінної типу URLRequest. Дайємо йому нашу основну URL-адресу та додаємо конкретний шлях, який будемо використовувати.
2. Встановлюємо httpMethod запиту рівним такому, як в EndPoint.
3. Створюємо блок "do-try-catch", оскільки наші кодери можуть видавати помилку. Створюючи один великий блок "do-try-catch", нам не потрібно створювати окремий блок для кожної спроби.
4. Проходимо по route.task
5. Залежно від завдання, викликаємо відповідний кодер.

```
fileprivate func buildRequest(from route: EndPoint) throws -> URLRequest {

    var request = URLRequest(url: route.baseURL.appendingPathComponent(route.path),
                             cachePolicy: .reloadIgnoringLocalAndRemoteCacheData,
                             timeoutInterval: 10.0)

    request.httpMethod = route.httpMethod.rawValue
    do {

        if let headers = route.headers{
            self.addAdditionalHeaders(headers, request: &request)
        }
        switch route.task {
        case .request:
            request.setValue("application/json", forHTTPHeaderField: "Content-Type")
        case .requestParameters(let bodyParameters,
                                let bodyEncoding,
                                let urlParameters):

            try self.configureParameters(bodyParameters: bodyParameters,
                                         bodyEncoding: bodyEncoding,
                                         urlParameters: urlParameters,
                                         request: &request)

        case .requestParametersAndHeaders(let bodyParameters,
                                          let bodyEncoding,
                                          let urlParameters,
                                          let additionalHeaders):

            self.addAdditionalHeaders(additionalHeaders, request: &request)
            try self.configureParameters(bodyParameters: bodyParameters,
                                         bodyEncoding: bodyEncoding,
                                         urlParameters: urlParameters,
                                         request: &request)

        }
        return request
    } catch {
        throw error
    }
}
```

Рис. 3.14 Побудований Роутер

Також у випадку коли потрібно додати додаткові заголовки в реквест створено функцію.

```
    fileprivate func addAdditionalHeaders(_ additionalHeaders: HTTPHeaders?, request: inout URLRequest) {
        guard let headers = additionalHeaders else { return }
        for (key, value) in headers {
            request.setValue(value, forHTTPHeaderField: key)
        }
    }
}
```

Рис. 3.15 Реалізація додаткових заголовків

3.4.3 Підтримка різних статусів

Важливою частиною є підтримка різних типів відповідей реквесту. Різні статуси відповіді з BackEnd повинні бути оброблені

```
class Network {
    static func handleNetworkResponse(_ response: HTTPURLResponse) -> Result<String>{
        switch response.statusCode {
        case 200...299: return .success
        case 401...403: return .outDatedToken
        case 404...500: return .failure(NetworkResponse.authenticationError.rawValue)
        case 501...599: return .failure(NetworkResponse.badRequest.rawValue)
        case 600: return .failure(NetworkResponse.outdated.rawValue)
        default: return .failure(NetworkResponse.failed.rawValue)
        }
    }
}


enum NetworkResponse:String {
    case success
    case authenticationError = "You need to be authenticated first."
    case badRequest = "Bad request"
    case outdated = "The url you requested is outdated."
    case failed = "Network request failed."
    case noData = "Response returned with no data to decode."
    case unableToDecode = "We could not decode the response."
    case oldAccessToken = "Should refresh access_token"
    case serverRip = "Server is RIP"
}
```

Рис. 3.15 Реалізація підтримки статусів реквесту

3.5 API

Наприклад, для сторінки персонального кабінету потрібні такі типи реквестів:

- Реквест на Особистий профіль
- Реквест на профіль Іншого користувача
- Підписатися/Відписатися від користувача

Тому створюємо файл “UserEndPoint“, в якому створюємо enum “UserApi”, де прописуємо реквести.

```
public enum UserApi {
    case me
    case getUser(id: Int)
    case subscribe(id: Int)
    case getUsersBySearch(query : String)
}
```

Рис. 3.16 Реалізація UserApi

Далі ми повинні визначити тип BackEnd: тестовий чи бойовий.

```
var environmentBaseURL : String {
    switch EventManager.environment {
    case .production: return "https://events-core.herokuapp.com/prod"
    case .qa: return "https://events-core.herokuapp.com/qa"
    case .staging: return "https://events-core.herokuapp.com/stage"
    }
}

var baseURL: URL {
    guard let url = URL(string: environmentBaseURL) else { fatalError("baseURL could not be configured.")}
    return url
}
```

Рис. 3.16 Додавання типу BackEnd

Прописуємо шлях кожного із реквестів та вказуємо який тип HTTP метода буде використовуватися.

```
var path: String {
    switch self {
    case .me:
        return "/api/users/me"
    case .getUser(let id):
        return "/users/\(id)"
    case .subscribe(let id):
        return "/subscribe/\(id)"
    case .getUsersBySearch:
        return "/api/users/find"
    }
}

var httpMethod: HTTPMethod {
    switch self {
    case .me:
        return .get
    case .getUser:
        return .get
    case .subscribe:
        return .put
    case .getUsersBySearch:
        return .get
    }


}
```

Рис. 3.17 Додавання шляху та метода реквеста

Вказуємо параметри до вказаних реквестів та вказуємо тип кодування.

```
var task: HTTPTask {

    switch self {
    case .me:
        return .requestParameters(bodyParameters: nil, bodyEncoding: .urlEncoding, urlParameters: nil)
    case .getUser:
        return .requestParameters(bodyParameters: nil, bodyEncoding: .urlEncoding, urlParameters: nil)
    case .subscribe:
        return .requestParameters(bodyParameters: nil, bodyEncoding: .urlEncoding, urlParameters: nil)
    case .getUsersBySearch(let query):
        return .requestParameters(bodyParameters: nil, bodyEncoding: .urlEncoding, urlParameters: ["q":query])
    }
}

var headers: HTTPHeaders? {
    return ["Authorization":TokenManager.getToken() ?? ""]
}
}
```

Рис. 3.17 Додавання параметрів реквеста

```
let router = Router<EventApi>()
router.request(EventApi.getEvents(page: page, size: size)) { data, response, error in
```

Рис. 3.18 Результат

3.6 Реєстрація та логін

У комп'ютерній безпеці логін - це процес, за допомогою якого людина отримує доступ до системи шляхом ідентифікації та автентифікації. Користувацькі дані, як правило, є деякою формою "ім'я користувача" та відповідним "паролем. У додатку користувачу дається вибір зайти у систему, або зареєструватися в системі. Після того, як додаток загрузив всі потрібні для існування данні дивиться на статус користувача. Якщо статус входу в систему користувача є false, то він потрапляє на LogIn.storyboard.

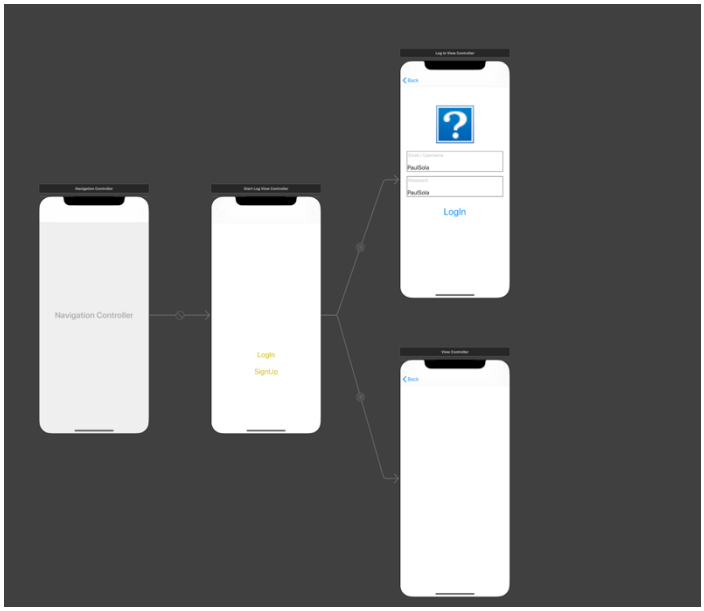


Рис. 3.19 Ієрархія LogIn.storyboard

Користувач може зареєструватися у системі, або увійти, якщо він вже зареєстрований.

Користувач водить Логін та пароль і відправляє запит на BackEnd. Якщо BackEnd знайшов користувача у базі данних – то у відповіді прийде:

- Access Token
- Refresh Token

Кожен користувач має свій унікальний access token, який дозволяє йому користуватися додатком.

Кожен запит містить у собі цей унікальний ключ, який шифрує данні та

ідентифікує користувача. Але access token може обновитися (це зроблено для того, щоб зловмисники не змогли дістати персональні данні). Кожен запит приймає відповідь з BackEnd. Однією відповіддю з BackEnd є статус 403, який означає, що access token обновився. Саме тут нам знадобиться refresh token. Якщо у будь-якому запиті відповіддю є статус 403, то ми повинні спочатку відправити на BackEnd запит з refresh token та дочекатися відповіді з оновленим access token та спробувати зробити минулий запит ще раз.

У функції “parse” реалізована логіка оновлення токену. Статус відповіді може бути:

- Успішний
- Невдалий
- Потрібно оновити токен

При успішній відповіді ми декодуємо данні та відображаємо відповідну сторінку.

Якщо BackEnd надіслав статус “невдалий”, то ми повинні відобразити сторінку з описом проблеми, або попросити зачекати. Коли приходить статус “Потрібно оновити токен” ми зберігаємо локально посилання на запит до BackEnd у іншому потоці спробуємо оновити токен. При успішному оновленні токену ми повинні зробити той самий запит, використовуючи збережене посилання. При неуспішному оновленні токену – користувачу відобразиться відповідна сторінка з описом проблеми.

```
class JSONParser {

    typealias result<T> = (ParseResult<T>) -> Void

    static func parse<T: Decodable, EndPoint : EndPointType>(of type: T.Type,
                                  api : EndPoint,
                                  routerCompletion: RequestCompletion,
                                  completion: @escaping result<T>) {


        func shouldRefresh(){
            let authManager = AuthManager()

            func tryRequestAgain(){
                if TokenManager.getTokenStatus() {
                    let router = Router<EndPoint>()
                    router.request(api) { (data, response, error) in
                        parseAfterRefresh(of: T.self, routerCompletion: (data, response, error), completion: { (result) in
                            switch result {
                            case .failure(let error):
                                completion(.failure(error))
                            case .success(let object):
                                completion(.success(object))
                            }
                        })
                    }
                    print("Try Request again") }
                else {completion(.failure("Token is not valid"))}
            }

            if AuthManager.getRefreshStatus() != true{
                //AuthManager.setRefreshStatus(true)
                authManager.refreshToken() { (error) in
                    AuthManager.setRefreshStatus(false)
                    error == nil ? tryRequestAgain(): completion(.failure(error!))
                }
            }
            else{
                DispatchQueue.main.async {
                    while AuthManager.getRefreshStatus() == true {}
                    print("waited")
                    tryRequestAgain()
                }

            }
        }
```

Рис. 3.20 Реалізація оновлення токену

3.7 Головний контролер

Якщо користувач успішно увійшов в систему, то побачить головний контролер, який розділений на 4 частини:

- Головна сторінка Подій
- Пошук Подій та Інших користувачів
- Карта Подій
- Персональна Сторінка

Кожна частина містить свою ієрархію контролерів. Головний контролер зберігає посилання на файли типу .storyboard, які у свою чергу містять ієрархію контролерів та представлень. Файл типу storyboard повинен містити “вхідну точку”, щоб програма розуміла з якого контролера потрібно починати своє представлення.

У додатку реалізовано контролер UINavigationViewController, який є стеком для контролерів, які ми будемо додавати в нього.

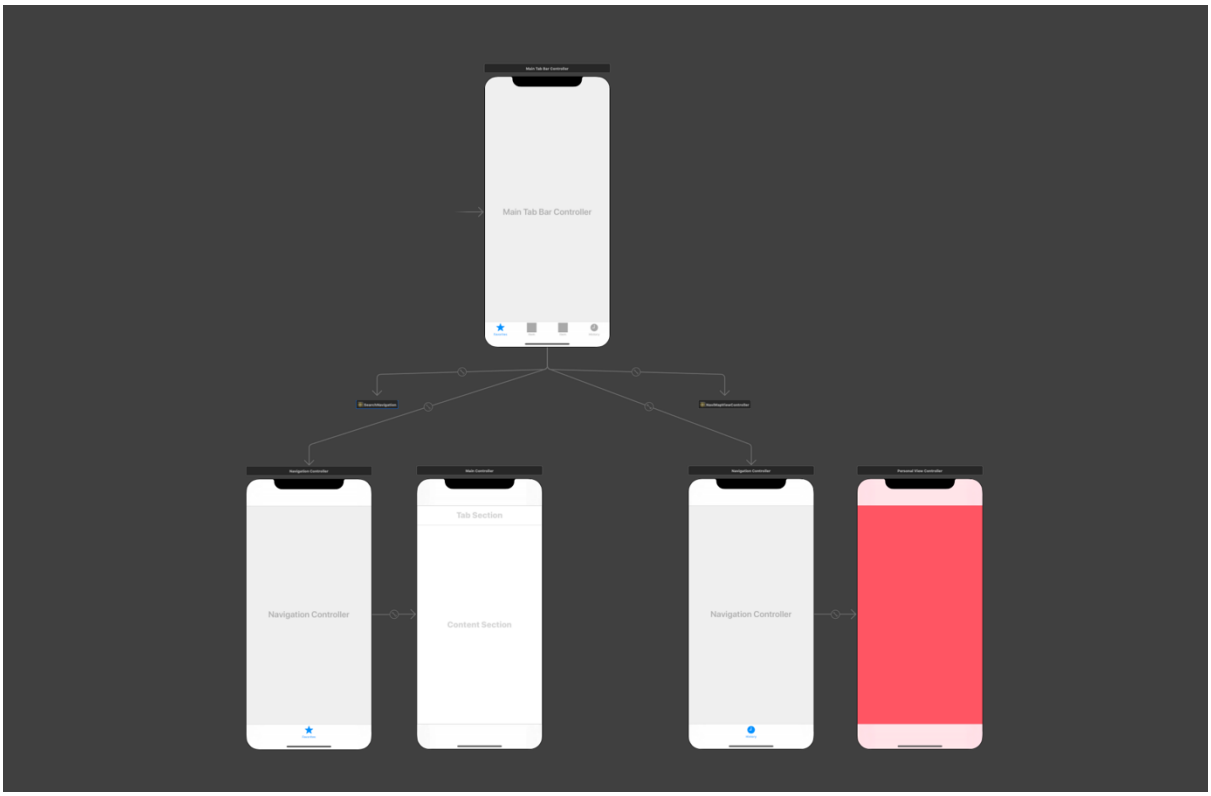


Рис. 3.21 Реалізація ієрархії головного контролера

ВИСНОВКИ ДО РОЗДІЛУ 3

Взаємодія з BackEnd вдалася:

- протокол-орієнована
- проста у використанні
- проста у виконанні
- Безпечна для типів даних

Використовувати даний код дуже зручно, так як створення будь-якого запиту на BackEnd тепер займає лише 2 строки коду. Дуже легко конфігурувати парамети різних типів. Це вдалося за допомогою гнучкого Encoder, який розпізнає тип параметру та вставляє потрібну кодировку. Реалізація декодування відповіді з BackEnd розроблена таким чином, потрібно вказати тип об’єкту, і програма самостійно декодує всі параматри та необхідні данні. Данна реалізація має можливість конфігурувати стадії BackEnd, що значно полегшує розробку та тестування. На стадії проєктування потрібно вирішити які стадії BackEnd будуть використовуватися, і таким чином можна динамічно їх змінювати та тестувати. Розроблена система фільтрації помилок. Якщо виникла якась помилка – система зробить повторний запит, і якщо знову прийде помилка – система сповістить користувача про невдалий запит.

Також за безпечність данних відповідає токен, який оновлюється з константою періодичністю, що запобігає викраденню данних користувача. Архітектура додатку вдалась, слідуючи всіма принципам SOLID : гнучкою, легко розширюваною, зрозумілою.

РОЗДІЛ 4.
ЕКСПЕРИМЕНТ І ТЕСТУВАННЯ

4.1 Головна сторінка

Головна сторінка містить список актуальних подій.

Реалізована “нескінченна прокрутка ”, який дозволяє загрузити тільки перші 5 подій, а не всі, які є на BackEnd та підгружати нові події коли треба.

“Нескінченна прокрутка” дозволяє користувачам поступово завантажувати вміст. Додаток завантажує деякі початкові дані, а потім завантажує решту інформації, коли користувач досягне нижньої частини видимого вмісту.

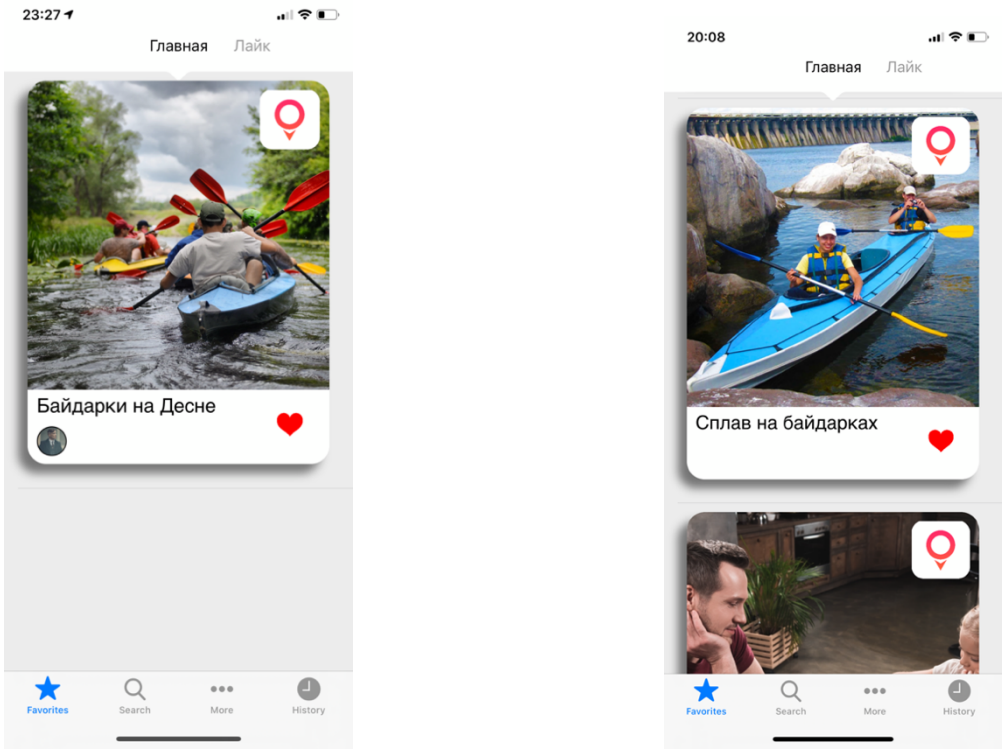


Рис. 4.1 Інтерфейс головної сторінки

Структура події:

- ID
- Назва
- Опис
- Дата проведення події
- Час проведення події

- Координати проведення події
- Картинка події
- Автор події
- Користувачі, які будуть на події
- Статус підтвердження присутності на події

У користувача є можливість “лайкнути” подію, що автоматично збереже цю подію у “сподобанні події”, це зроблено для того щоб у користувача була можливість стежити за подіями, які йому сподобалися. Під назвою події у користувача відображається аватарка автора події. Реалізована кнопка, яка дає можливість користувачу перейти на карту у додатку та подивитися де буде знаходитись ця подія.

Якщо користувач зайде на подію, то зможе отримати більш детальну інформацію.

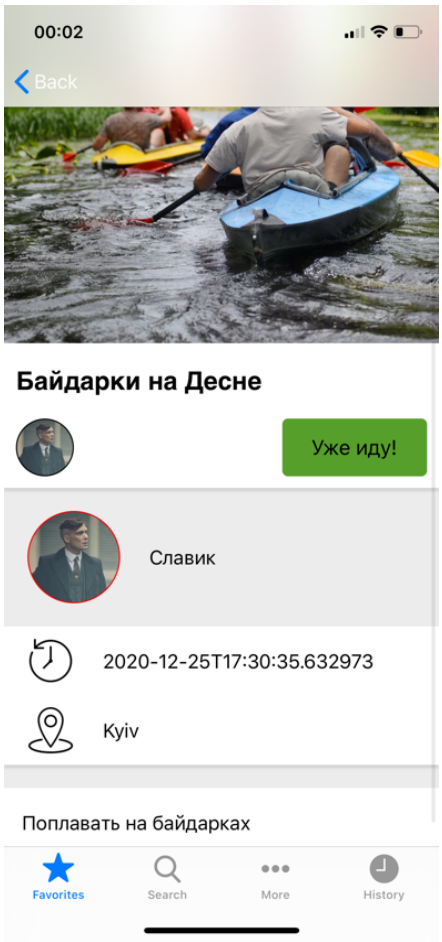


Рис. 4.2 Інтерфейс сторінки деталей події

Під назвою події відображаються аватарки користувачів, які йдуть на подію.

Якщо користувачів більше трьох, то відображається тільки перші 3 аватарки та з’являється відображення кількості користувачів, які йдуть на подію. У користувача є можливість підтвердити свою присутність на події. Статус присутності змінюється та наступного разу коли користувач перейде на цю подію – він побачить актуальний статус та інші користувачі побачать його у списку присутніх на цій події. Далі відображається аватар та нікнейм автора. У користувача є можливість перейти на сторінку автора. Також користувач зможе отримати інформацію про місце та час проводження події.

4.2 Сторінка пошуку подій та користувачів

Просто натисніть на панель пошуку, щоб почати пошук. Ви можете шукати:

- Події
- Інших користувачів

Результати пошуку ґрунтуються на різних факторах, включаючи людей, яких ви стежите, які вам подобаються івенти, місце події і т.п.

Всі події, якій вже пройшли зберігаються у базі данних. Тобто користувач зможе знайти будь-які події, навіть якщо вони вже закінчилися.

Також користувач зможе знайти інших користувачів. Та, якщо натиснути на користувача – можна перейти до його профілю та підписатися на нього.

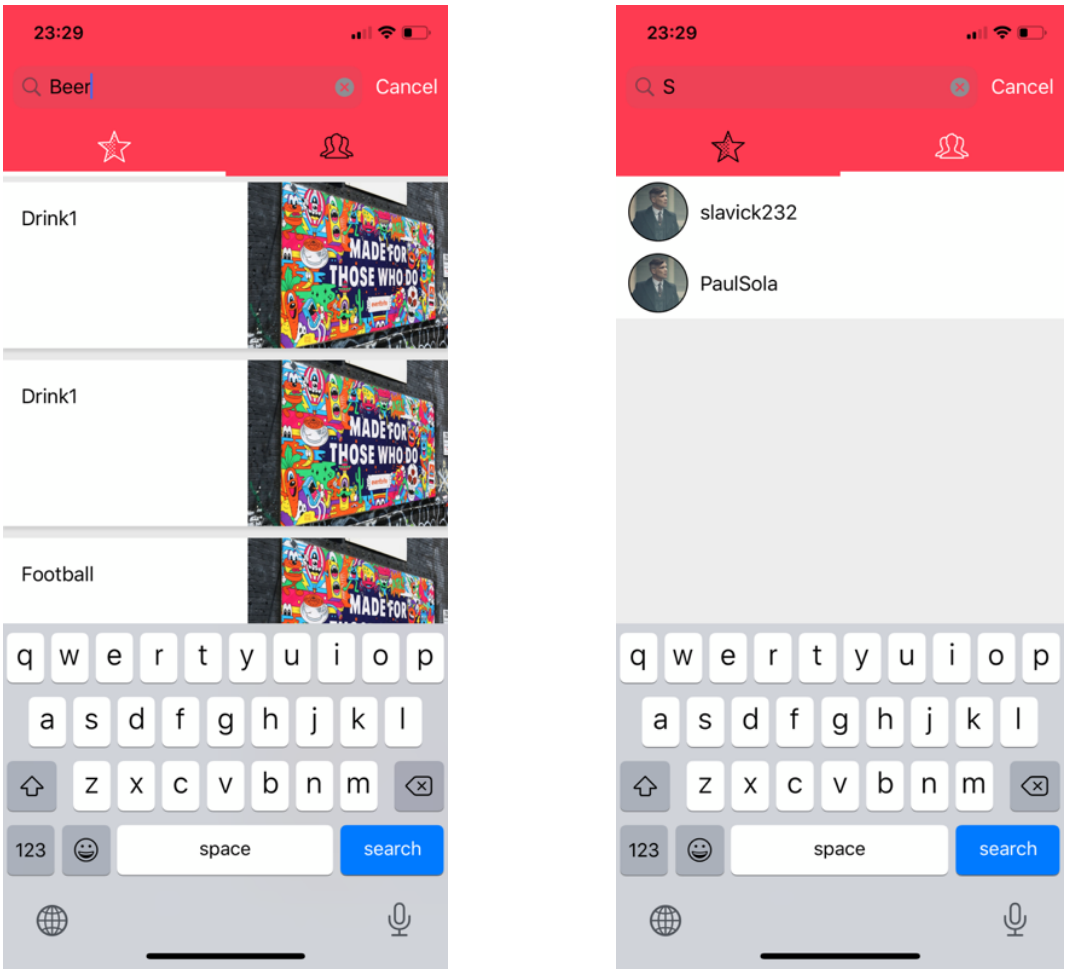


Рис. 4.3 Інтерфейс сторінки деталей події

4.3 Сторінка подій на карті

Дуже зручно шукати події на карті, так як подій поблизу дуже багато, а за допомогою карти можна подивитися актуальні події у конкретному регіоні. Також це дуже зручно саме тоді, коли користувач планує подорож до іншої країни. Він зможе перейти на карту та подивитися події, наприклад, у Європі. Та спланувати свою подорож до Європи саме на цю подію.

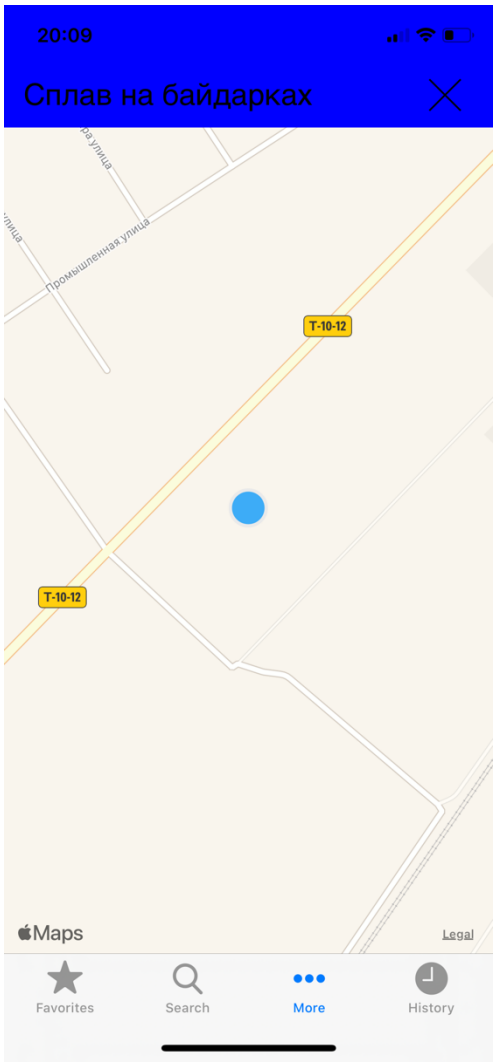


Рис. 4.4 Інтерфейс події на карті

Якщо користувач захоче подивитися події, які знаходяться саме, наприклад, у Києві – він зможе віддалити карту на регіон Києва. Всі події, які знаходяться у цьому регіоні зберуться у “кластер” (на карті користувач побачить одну точку з кількістю івентів у цьому “кластері”). Коли користувач скролить карту – додаток автоматично загружає події у полі видимості карти. Це дуже зручно, так як нема необхідності загружати всі події, які є на карті. Важливі тільки ті події, які знаходяться у полі видимості даного регіону, на якому знаходиться користувач. Але у нього є можливість перемістити карту у інший регіон, і одразу побачить інші події.

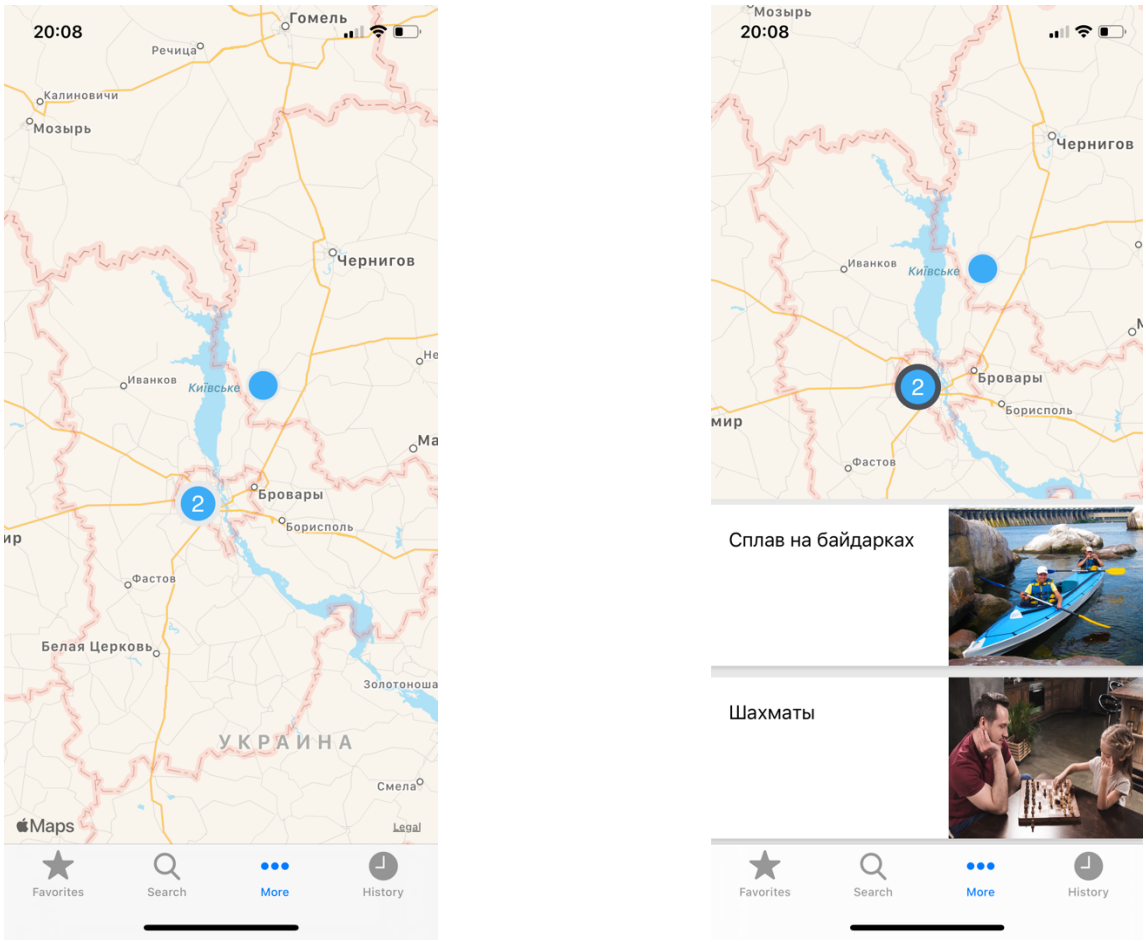


Рис. 4.5 Кластер події на карті

Натиснувши на “кластер”, користувач зможе переглянути події та перейти до конкретної події.

4.4 Сторінка користувача

Користувач має свою сторінку, на якій відображається його:

- Аватар
- Ім’я
- Нікнейм
- Кількість підписників
- Кількість підписок

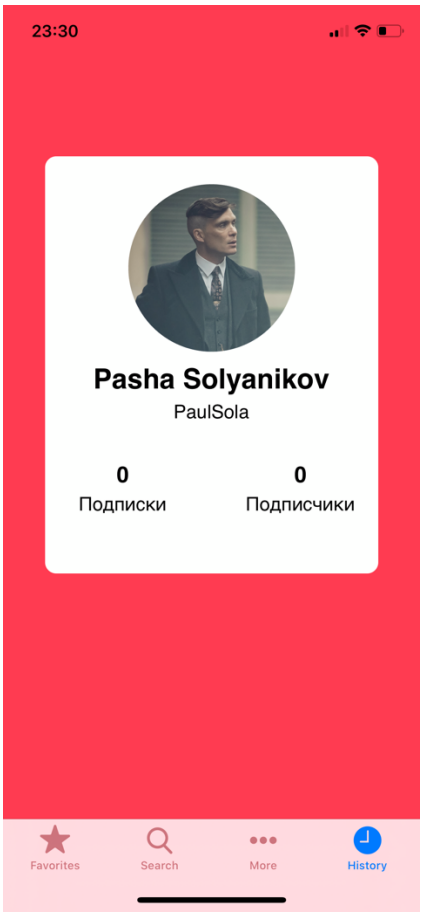


Рис. 4.6 Інтерфейс сторінки користувача

ВИСНОВКИ ДО РОЗДІЛУ 4

Розроблена система функціонує швидко. Додатком користуватися зручно. Інтерфейс простий та інтуїтивно зрозумілий. Анімації контролерів та переглядів працюють плавно, без преривань. Взаємодія з BackEnd працює швидко, без помилок. Безпечність даних користувача забезпечується автоматичним оновленням токену, який буде слугувати до наступного сеансу.

ВИСНОВКИ

Сьогодні соціальні мережі стали невід'ємною частиною життя кожного. Щороку з'являються все більше різноманітних соціальних мереж, які слугують для віртуального спілкування. Саме це є основною проблемою сьогодення – відсутність живого спілкування. Тому створення додатку, який буде стимулювати людей реально спілкуватися та проводити час разом є дуже важливим. Одним із головних факторів при розробці мобільного додатку була гнучкість архітектури системи. Світ дуже стрімко змінюється, саме тому потрібно завжди бути готовим перероблювати систему таким чином, щоб вона була повністю пристосованою до її користувачів. Головна сторінка відображає актуальні події в тій місцевості, в якій користувач знаходиться. Користувач має змогу швидко перейти на карту подій та подивитися її місцезнаходження. Користувач має можливість подивитися детальну інформацію про цю подію, а саме :

- Автора події
- Інших користувачів, які будуть присутні
- Час проведення
- Точне місцезнаходження
- Опис події

Це дає змогу користувачу вирішити, чи піде він на цю подію. Користувач має змогу переглянути всі створені події на карті. Це дуже спрощує пошук подій, якщо пріоритетом пошуку є місцезнаходження події. Також у користувача є можливість пошуку подій по ключовим словам. Пошук видає всі події, у яких частина назви чи опису співпадає з пошуковим ключовим словом. Також користувач може шукати та знаходити інших користувачів. Для цього потрібно зайти до пошуку користувачів. У користувача є власна сторінка, де він може слідкувати за своїми підписками та підписниками.

ПЕРЕЛІК ПОСИЛАНЬ

1. The Art of Social Media: Power Tips for Power Users Kawasaki, Guy, Peg Fitzpatrick / 2014
2. Social Media Explained / Mark W. Schaefer / 2018
3. The Social Media Bible: Tactics, Tools, and Strategies / Safko, Lon / 2009
4. The Power of Visual Storytelling: How to Use Visuals, Videos, and Social / Ekaterina Walter, Jessica Gioglio. / 2014
5. Data Structures & Algorithms in Swift / Kelvin Lau and Vy Vincent Ngo. / 2017
6. Mastering Swift 5: Deep Dive Into the Latest Edition of the Swift / Jon Hoffman / 2019
7. Beginning IOS 13 & Swift App Development / Greg Lim / 2019
8. Fundamentals with Swift / Matt Neuburg / 2015
9. Advanced Swift / Airspeed Velocity and Chris Eidhof / 2016
10. Functional Swift / Chris Eidhof and Florian Kugler 2014
11. Design Patterns by Tutorials / Jay Strawn and Joshua Greene / 2018
12. Advanced IOS App Architecture / Josh Berlin and Rene Cacheaux / 2019
13. IOS Apprentice: Beginning IOS Development with Swift / Matthijs Hollemans / 2013
14. Learning Swift: Building Apps for MacOS, IOS, and Beyond / 2017.
15. User interface for your iOS (UIKit) [Електронний ресурс]. – Apple Developer, 2019 - Режим доступу - https://developer.apple.com/documentation/uikit -
16. Using Network[Електронний ресурс]. – Apple Developer, 2019 - Режим доступу : https://developer.apple.com/documentation/network
17. Start Developing iOS Apps (Swift) [Електронний ресурс]. – Apple documentation archive, 2018 Режим доступу : https://developer.apple.com/library/archive/referencelibrary/GettingStarted/DevelopiOSAppsSwift/

18. MapKit Tutorial: Getting Started [Електронний ресурс]. – raywenderlich, 2018
- Режим доступу : https://www.raywenderlich.com/7738344-mapkit-tutorial-getting-started

ДОДАТОК А. Код програми.

1. EventManager

```
import Foundation

struct EventManager {

   static let environment : NetworkEnvironment = .staging
   static let EventAPIKey = ""
   let router = Router<EventApi>()

   func getEvents( page : Int, size : Int, completion: @escaping (_ event: [Event]?,_ error:
String?)->()){

      let api = EventApi.getEvents(page: page, size: size)
       router.request(api) { data, response, error in

          JSONParser.parse(of: [SimpleEvent].self, api: api, routerCompletion: (data, response,
error), completion: { (result) in

                switch result {
                case .success(let events) :
                   completion(events, nil)
                case .failure(let error):
                   completion(nil, error)
                }
             })
       }
   }

   func getEventsOnMapByLocation(_ location : BoundsLocation, completion: @escaping (_
event: [Event]?,_ error: String?)->()){
      let api = EventApi.getMapBoundsEvents(location: location)
      router.request(api) { (data, response, error) in

          JSONParser.parse(of: [SimpleEvent].self, api: api, routerCompletion: (data, response,
error), completion: { (result) in

             switch result {
             case .success(let events) :
                completion(events, nil)
             case .failure(let error):
                completion(nil, error)
             }
          })
       }

   }

   func getEventsBySearch(query : String, completion: @escaping (_ events: [Event]?,_ error:
String?)->()){
```

```
        let api = EventApi.getEventsBySeach(query: query)
        router.request(api) { (data, response, error) in

            JSONParser.parse(of: [SimpleEvent].self, api: api, routerCompletion: (data, response,
error), completion: { (result) in

                switch result {
                case .success(let events) :
                    completion(events, nil)
                case .failure(let error):
                    completion(nil, error)
                }
            })
        }
    }

    func visitEvent(eventId : Int ,completion: @escaping (_ error: String?)->()){
        let api = EventApi.visitEvent(id: eventId)
        router.request(api) { (data, response, error) in
            if let response = response as? HTTPURLResponse{
                let result = Network.handleNetworkResponse(response)

                switch result {
                case .success:
                    completion(nil)
                case .failure(let error):
                    completion(error)
                case .outDatedToken:
                    completion("outDatedToken")
                }
            }

        }
    }


    func postEvent(title:String, desc:String,completion: @escaping (_ error: String?)->()){
        router.request(.postEvent(title: title, description: desc)) { data, response, error in
            if error != nil {
                completion("Please check your network connection.")
            }

            if let response = response as? HTTPURLResponse {
                let result = Network.handleNetworkResponse(response)
                switch result {
                case .success:
                    completion(nil)
                case .failure(let networkFailureError):
                    completion( networkFailureError)
                case .outDatedToken:
                    print("Should update token")
                }
```

```
        }
      }
    }
}
```

2. AuthManager

```
import Foundation

struct AuthManager {

    static let environment : NetworkEnvironment = .staging
    static let EventAPIKey = ""
    let router = Router<AuthApi>()

    private static var isRefreshing : Bool = false

    static func getRefreshStatus() -> Bool {
        return isRefreshing
    }
    static func setRefreshStatus(_ status : Bool){
        isRefreshing = status
    }

    func logInUser(firstTime : Bool = true, userIdentifer:String, password:String,completion:
@escaping (_ token : Token?, _ refreshToken: Token?, _ error: String?)->()){
        router.request(.logInUser(userIdentifier: userIdentifer, password: password)) { data,
response, error in
            if error != nil {
                completion(nil, nil, "Please check your network connection.")
            }

            if let response = response as? HTTPURLResponse {
                let result = Network.handleNetworkResponse(response)
                switch result {
                case .success:
                    guard let responseData = data else {
                        completion(nil, nil, NetworkResponse.noData.rawValue)
                        return
                    }
                    do {
                        let json = try JSONSerialization.jsonObject(with: responseData, options:
.mutableContainers) as? [String: Any]
                        if let token = json?["accessToken"] as? String,
                            let refreshToken = json?["refreshToken"] as? String{
                            completion(token, refreshToken, nil)
                        }
                        else{
                            completion(nil, nil, NetworkResponse.unableToDecode.rawValue)
```

```
                }

            }catch {
                print(error)
                completion(nil, nil, NetworkResponse.unableToDecode.rawValue)
            }

        case .failure(let networkFailureError):
            completion(nil, nil, networkFailureError)
        case .outDatedToken:
            completion(nil, nil, NetworkResponse.authenticationError.rawValue)
        }
    }
  }
}


func refreshToken(completion: @escaping ( _ error: String?)->()){

    TokenManager.setTokenStatus(status: false)
    AuthManager.setRefreshStatus(true)
    router.request(.refreshToken) { data, response, error in
        if error != nil {
            completion("Please check your network connection.")
        }

        if let response = response as? HTTPURLResponse {
            print(response)
            let result = Network.handleNetworkResponse(response)
            switch result {
            case .success:
                guard let responseData = data else {
                    completion(NetworkResponse.noData.rawValue)
                    return
                }
                do {
                    let json = try JSONSerialization.jsonObject(with: responseData, options:
.mutableContainers) as? [String: Any]
                    if let token = json?["accessToken"] as? String,
                        let refreshToken = json?["refreshToken"] as? String{
                        print(AuthManager.getRefreshStatus())
                        TokenManager.setToken(token: token)
                        TokenManager.setTokenStatus(status: true)
                        TokenManager.setRefreshToken(refreshToken: refreshToken)
                        completion(nil)
                    }
                    else{
                        completion(NetworkResponse.unableToDecode.rawValue)
                    }

                }catch {
                    print(error)
```

```
                completion(NetworkResponse.unableToDecode.rawValue)
            }

            case .failure(let networkFailureError):
                completion(networkFailureError)
            case .outDatedToken:
                completion(NetworkResponse.serverRip.rawValue)
            }
        }
    }
  }
}
```

3. JsonParcer

```
class JSONParser {

    typealias result<T> = (ParseResult<T>) -> Void

    static func parse<T: Decodable, EndPoint : EndPointType>(of type: T.Type,
                                api : EndPoint,
                                routerCompletion: RequestCompletion,
                                completion: @escaping result<T>) {


        func shouldRefresh(){
            let authManager = AuthManager()

            func tryRequestAgain(){
                if TokenManager.getTokenStatus() {
                    let router = Router<EndPoint>()
                    router.request(api) { (data, response, error) in
                        parseAfterRefresh(of: T.self, routerCompletion: (data, response, error),
completion: { (result) in
                            switch result {
                            case .failure(let error):
                                completion(.failure(error))
                            case .success(let object):
                                completion(.success(object))
                            }
                        })
                    }
                    print("Try Request again") }
                else {completion(.failure("Token is not valid"))}
            }

            if AuthManager.getRefreshStatus() != true{
                //AuthManager.setRefreshStatus(true)
```

```
            authManager.refreshToken() { (error) in
                AuthManager.setRefreshStatus(false)
                error == nil ? tryRequestAgain(): completion(.failure(error!))
            }
        }
        else{
            DispatchQueue.main.async {
                while AuthManager.getRefreshStatus() == true {}
                print("waited")
                tryRequestAgain()
            }

        }
    }

        if routerCompletion.2 != nil {
            completion(.failure("Please check your network connection."))
        }
        if let response = routerCompletion.1 as? HTTPURLResponse {
            //print(response)
            let result = Network.handleNetworkResponse(response)
            switch result {
            case .success:
                guard let responseData = routerCompletion.0 else {
                    completion(.failure(NetworkResponse.noData.rawValue))
                    return
                }
                do {

                    print( try JSONSerialization.jsonObject(with:
                        responseData, options: []))

                    let apiResponse: T = try JSONDecoder().decode(T.self, from: responseData)
                    completion(.success(apiResponse))
                }catch {

                    completion(.failure(NetworkResponse.unableToDecode.rawValue +
error.localizedDescription))
                }
            case .failure(let networkFailureError):
                completion(.failure(networkFailureError))
            case .outDatedToken:
                print(AuthManager.getRefreshStatus())
                shouldRefresh()
            }
        }
    }

    private static func customParse(routerCompletion: RequestCompletion,
    completion: @escaping(_ result: Any?,_ error: String?)->()) {
        if routerCompletion.2 != nil {
            completion(nil, "Please check your network connection.")
```

```
        }
        if let response = routerCompletion.1 as? HTTPURLResponse {
            let result = Network.handleNetworkResponse(response)
            switch result {
            case .success:
                guard let responseData = routerCompletion.0 else {
                    completion(nil, NetworkResponse.noData.rawValue)
                    return
                }
                do {
                    let apiResponse = try JSONSerialization.jsonObject(with:
                        responseData, options: [])
                    completion(apiResponse , nil)

                }catch {
                    completion(nil, NetworkResponse.unableToDecode.rawValue)
                }
            case .failure(let networkFailureError):
                completion(nil ,networkFailureError)
            case .outDatedToken:
                completion(nil, "RIP")
            }
        }
    }

    private static func parseAfterRefresh<T: Decodable>(of type: T.Type,
                                    routerCompletion: RequestCompletion,
                                    completion: @escaping result<T>) {
        if routerCompletion.2 != nil {
            completion(.failure("Please check your network connection."))
        }
        if let response = routerCompletion.1 as? HTTPURLResponse {
            let result = Network.handleNetworkResponse(response)
            switch result {
            case .success:
                guard let responseData = routerCompletion.0 else {
                    completion(.failure(NetworkResponse.noData.rawValue))
                    return
                }
                do {
                    let apiResponse: T = try JSONDecoder().decode(T.self, from: responseData)
                    completion(.success(apiResponse))
                }catch {
                    completion(.failure(NetworkResponse.unableToDecode.rawValue))
                }
            case .failure(let networkFailureError):
                completion(.failure(networkFailureError))
            case .outDatedToken:
                completion(.failure("RIP"))
            }
        }
```

```
    }
}
```

EventAPI

```
import Foundation

//add func-requests here
public enum EventApi {
    case getEvents(page: Int, size : Int)
    case postEvent(title: String, description: String)
    case getMapBoundsEvents(location : BoundsLocation)
    case getEventsBySeach(query : String)
    case visitEvent(id : Int)
}

extension EventApi: EndPointType {

    var environmentBaseURL : String {
        switch EventManager.environment {
        case .production: return "https://events-core.herokuapp.com"
        case .qa: return "https://events-core.herokuapp.com"
        case .staging: return "https://events-core.herokuapp.com"
        }
    }

    var baseURL: URL {
        guard let url = URL(string: environmentBaseURL) else { fatalError("baseURL could not
be configured.")}
        return url
    }

    var path: String {
        switch self {
        case .getEvents:
            return "/api/events"
        case .postEvent:
            return "/api/events/add"
        case .getMapBoundsEvents:
            return "/api/maps/events"
        case .getEventsBySeach(_):
            return "/api/events/find"
        case .visitEvent(let id):
            return "/api/events/visit/\(id)"
        }
    }
```

```
    var httpMethod: HTTPMethod {
        switch self {
        case .getEvents:
            return .get
        case .postEvent:
            return .post
        case .getMapBoundsEvents:
            return .get
        case .getEventsBySeach:
            return .get
        case .visitEvent:
            return .put
        }
    }

    var task: HTTPTask {

        switch self {
        case .getEvents(let page, let size):
            return .requestParametersAndHeaders(bodyParameters: nil, bodyEncoding:
.urlEncoding, urlParameters: ["page" : page, "limit" : size], additionHeaders: nil)

        case .postEvent(let title,let desc):
            return .requestParametersAndHeaders(bodyParameters: ["title":title, "description":desc],
bodyEncoding: .jsonEncoding, urlParameters: nil, additionHeaders: nil)

        case .getMapBoundsEvents(let location):
            let params = ["leftBotLatitude": location.leftBottomLatitude,
                          "leftBotLongitude": location.leftBottomLongitude,
                          "rightTopLatitude": location.rightTopLatitude,
                          "rightTopLongitude": location.rightTopLongitude]

            return .requestParameters(bodyParameters: nil, bodyEncoding: .urlEncoding,
urlParameters: params)

        case .getEventsBySeach(let query):
            return .requestParameters(bodyParameters: nil, bodyEncoding: .urlEncoding,
urlParameters: ["q": query])
        case .visitEvent:
            return .request
        }
    }

    var headers: HTTPHeaders? {
        return ["Authorization":TokenManager.getToken() ?? ""]
    }

}
```

MapViewController (Контролер карти перегляду)

```
class MapViewController: UIViewController {

    @IBOutlet weak var mapView: MKMapView!
    private var mapChangedFromUserInteraction = false
    lazy var bottomSheetVC : BottomAnimatedViewController = {
        let bottAnmVC = BottomAnimatedViewController()
        bottAnmVC.delegate = self
        return bottAnmVC
    }()

    lazy var aloneEventBar : AloneEventBar = {
        let alneEvBar = AloneEventBar(frame: CGRect(x: 0, y: -200, width: Device.width, height:
Device.statusBarHeight + 50))
        alneEvBar.delegate = self
        return alneEvBar
    }()

    enum MapState {
        case map
        case event
    }
    var mapState : MapState = .map
    var aloneEvent : Event? = nil

    var currentState: SlideOutState = .hidden

    var events : [Event] = [Event]()
    let eventsManager : EventManager = EventManager()
     let nonBackEventHelper = NonBackEvent()

    let regionRadius: CLLocationDistance = 500
    func centerMapOnLocation(location: CLLocation) {
        let coordinateRegion = MKCoordinateRegion(center: location.coordinate,
                                                  latitudinalMeters: regionRadius, longitudinalMeters:
regionRadius)
                                                   mapView.setRegion(coordinateRegion, animated: true)
    }

    @objc func changeMap(_ notification:Notification) {
        if let event = notification.userInfo?["event"] as? Event {
            mapState = .event
            removeAllAnnotations()
            showAloneEvent(event)
            mapView.addAnnotation(event)
            let initialLocation = CLLocation(latitude: event.coordinate.latitude, longitude:
event.coordinate.longitude)
            centerMapOnLocation(location: initialLocation)
        }

    }

    func  showAloneEvent(_ event : Event){
```

```
        aloneEvent = event
        aloneEventBar.setEvent(with: event)
        aloneEventBar.frame = CGRect(x: 0, y: 0, width: Device.width, height:
Device.statusBarHeight + 50)
    }

    func getEventsByLocation() {

//        print("location : \(getMapBounds(mapView!.visibleMapRect))")
//        removeAllAnnotations()
//        events = nonBackEventHelper.getEvents()
//        if events != nil{ mapView.addAnnotations(events!)}

        let mapBounds = getMapBounds(mapView!.visibleMapRect)
        eventsManager.getEventsOnMapByLocation(mapBounds) { (events, error) in
            if error != nil { print(error!)} else {
                DispatchQueue.main.async { [unowned self] in
                    //self.removeAllAnnotations() ; self.events = events!
                    let setOldEvents = self.events
                    var setNewEvents = events!



                    let eventsToStay: Array<Event> = setOldEvents.filter(setNewEvents.contains)
                    //setNewEvents.subtract(setOldEvents)
                    let result = setNewEvents.filter { !setOldEvents.contains($0) }
                    print(result, eventsToStay)
                    var count = 0
                    for annotation in self.mapView.annotations{
                        if let annEvent = annotation as? Event{
                            if !eventsToStay.contains(annEvent){
                                self.mapView.removeAnnotation(annotation)
                            }
                            count += 1
                        }

                    }
                    //print(self.mapView.annotations.count, count)
                    self.mapView.addAnnotations(Array(result))
                    self.events = events!
                }

            }
        }

    }

    private func configureMapController(){
        mapView.delegate = self
        view.addSubview(aloneEventBar)
        addBottomAnimatedView()
        registerAnnotationViewClasses()
```

```
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        NotificationCenter.default.addObserver(self, selector: #selector(changeMap(_:)), name:
.showEventOnMap, object: nil)
        configureMapController()

        let initialLocation = CLLocation(latitude: 50.4501, longitude: 30.5234)
        centerMapOnLocation(location: initialLocation)
        getEventsByLocation()

    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        navigationController?.setNavigationBarHidden(true, animated: animated)
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        navigationController?.setNavigationBarHidden(false, animated: animated)
    }

    private func registerAnnotationViewClasses() {
        mapView.register(EventAnnotationView.self, forAnnotationViewWithReuseIdentifier:
MKMapViewDefaultAnnotationViewReuseIdentifier)
        mapView.register(EventClusterView.self, forAnnotationViewWithReuseIdentifier:
MKMapViewDefaultClusterAnnotationViewReuseIdentifier)
    }
}

extension MapViewController: MKMapViewDelegate {

    func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) ->
MKAnnotationView? {

        if let cluster = annotation as? MKClusterAnnotation{
            return EventClusterView(annotation: cluster, reuseIdentifier:
EventClusterView.ReuseID) }
        else{
            return  EventAnnotationView(annotation: annotation, reuseIdentifier:
EventAnnotationView.ReuseID)}
    }

    func mapView(_ mapView: MKMapView, regionWillChangeAnimated animated: Bool) {
        if currentState == .expanded {
            hideView()
            for annotation in mapView.selectedAnnotations{
                mapView.deselectAnnotation(annotation, animated: true)
            }
```

```
        }
    }

    func mapView(_ mapView: MKMapView, didSelect view: MKAnnotationView) {

        if let cluster = view.annotation as? MKClusterAnnotation {
            let clusterEvents = cluster.memberAnnotations as? [Event]
            showView(with: clusterEvents!, isCluster: true)
        }
        else {
            let annotation = view.annotation as? Event
            showView(with: [annotation!], isCluster: false)
        }
    }

    func mapView(_ mapView: MKMapView, didDeselect view: MKAnnotationView) {
        hideView()
    }
    func mapView(_ mapView: MKMapView, regionDidChangeAnimated animated: Bool) {
        if mapViewRegionDidChangeFromUserInteraction() && aloneEvent == nil{
            getEventsByLocation()

        }
    }

    func removeAllAnnotations(){
        mapView.removeAnnotations(mapView!.annotations)
    }
}

extension MapViewController {


    enum SlideOutState {
        case hidden
        case expanded
    }

    func addBottomAnimatedView(){
        // 1- Init bottomSheetVC
        bottomSheetVC = BottomAnimatedViewController()

        // 2- Add bottomSheetVC as a child view
        self.addChild(bottomSheetVC)
        self.view.addSubview(bottomSheetVC.view)
        bottomSheetVC.didMove(toParent: self)

        // 3- Adjust bottomSheet frame and initial position.
        let height = view.frame.height
        let width  = view.frame.width
        bottomSheetVC.view.frame = CGRect(x: 0, y: self.view.frame.maxY, width: width, height:
height)
```

```
    }

    func showView(with events : [Event], isCluster : Bool){
        currentState = .expanded
        let tBheight : CGFloat = isCluster ? 300 : 150
        bottomSheetVC.updateView(with: events, tableViewHeight: tBheight)
        UIView.animate(withDuration: 0.3) {
            let yComponent = Device.safeHeight(with : self.view) + Device.statusBarHeight -
tBheight
            self.bottomSheetVC.view.frame = CGRect(x: 0, y: yComponent, width:
self.view.frame.width, height: self.view.frame.height)
        }
    }

    func hideView(){
        print("hideView")
        currentState = .hidden
        UIView.animate(withDuration: 0.3) {

            let yComponent = Device.height
            self.bottomSheetVC.view.frame = CGRect(x: 0, y: yComponent, width:
self.view.frame.width, height: self.view.frame.height)
        }
    }
}

extension MapViewController : MapBottomDelegate, MapAloneBarDelegate {


    func cancelBar() {
        aloneEvent = nil
        getEventsByLocation()
    }

    func showEvent(viewController: UIViewController) {
        self.navigationController?.pushViewController(viewController, animated: true)
    }

}
```

SearchViewController(контролер пошуку)

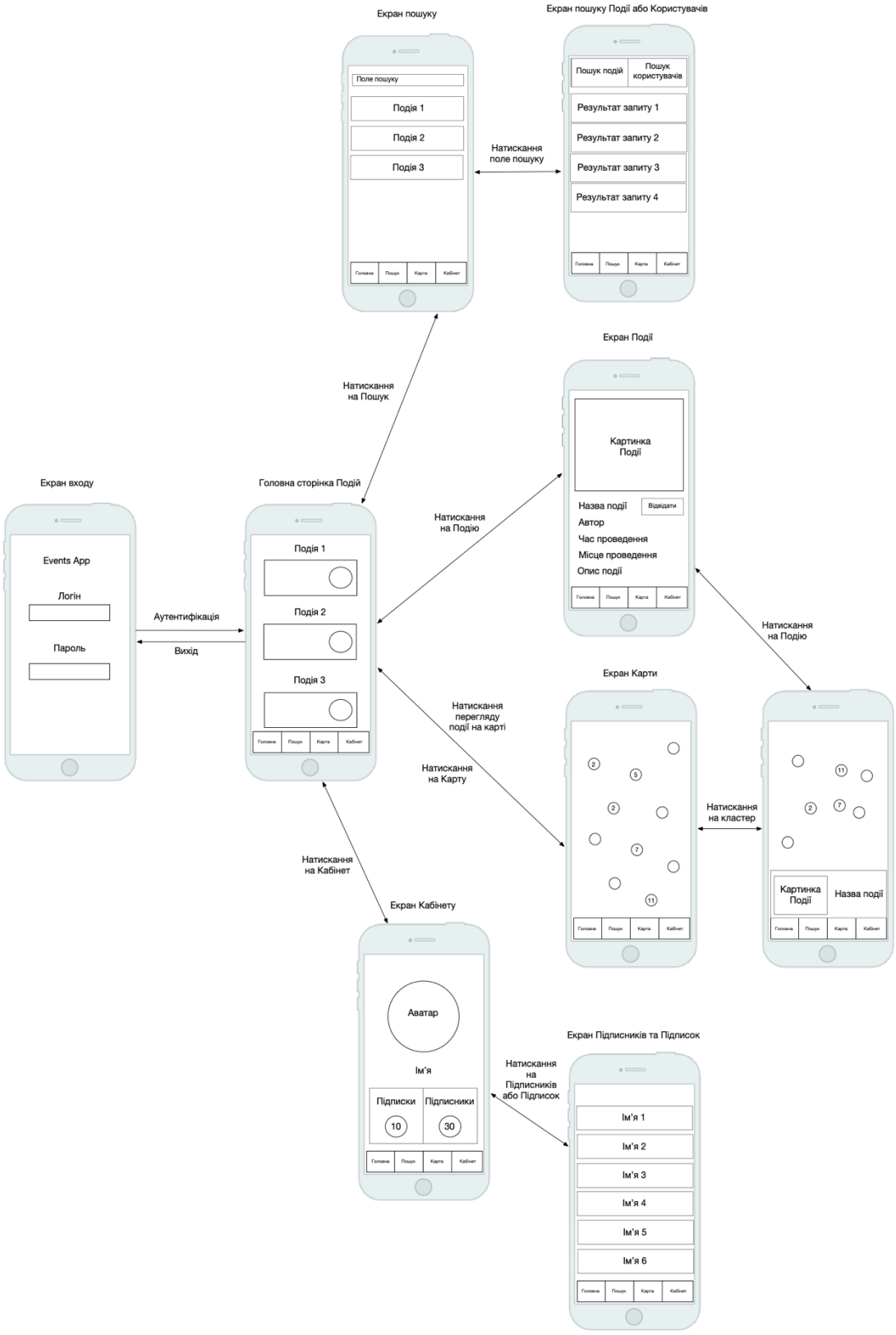
```
import UIKit


class SearchViewController: UIViewController, UICollectionViewDelegate,
UICollectionViewDataSource, UICollectionViewDelegateFlowLayout {

    enum SectionType {
        case event
        case people
```

```
    }
    var sectionTypes : [SectionType] = [.event, .people]
    lazy var currentSectionType : SectionType = sectionTypes.first!

        let layout: UICollectionViewFlowLayout = {
        let layout = UICollectionViewFlowLayout()
        layout.sectionInset = UIEdgeInsets(top: 0, left: 0, bottom: 0, right: 0)
        layout.itemSize = CGSize(width: 60, height: 60)
        return layout
    }()

    lazy var collectionView : UICollectionView = {
        let collView = UICollectionView(frame: self.view.frame, collectionViewLayout: layout)
        collView.translatesAutoresizingMaskIntoConstraints = false
        collView.backgroundColor =  colorLiteral(red: 0.9313134518, green: 0.9313134518, blue:
0.9313134518, alpha: 1)
        collView.register(EventTypeCollectionCell.self, forCellWithReuseIdentifier:
"EventTypeCollectionCell")
        collView.register(PeopleTypeCollectionCell.self, forCellWithReuseIdentifier:
"PeopleTypeCollectionCell")
        collView.isPagingEnabled = true
        self.view.addSubview(collView)
        return collView
    }()

    var numberOfItems = 1
    let titles : [String] = ["Events", "People"]
    let cellId = "cellIdCollection"
    var kpiTemplates = [KpiTemplate]()
    let tamplateHelper = CheckFeatures()

    lazy var menuBar : MenuBar = {
        let mb = MenuBar()
        mb.mainController = self
        mb.eventTemplates = kpiTemplates
        return mb
    }()

    func addMenuBar(){

        let redView = UIView()
        redView.backgroundColor = .red
        view.addSubview(redView)
        view.addConstraintsWithFormat(format: "H:|[v0]|", views: redView)
        view.addConstraintsWithFormat(format: "V:[v0(50)]", views: redView)

        view.addSubview(menuBar)
        view.addConstraintsWithFormat(format: "H:|[v0]|", views: menuBar)
        view.addConstraintsWithFormat(format: "V:[v0(50)]", views: menuBar)
        menuBar.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor).isActive =
true
```

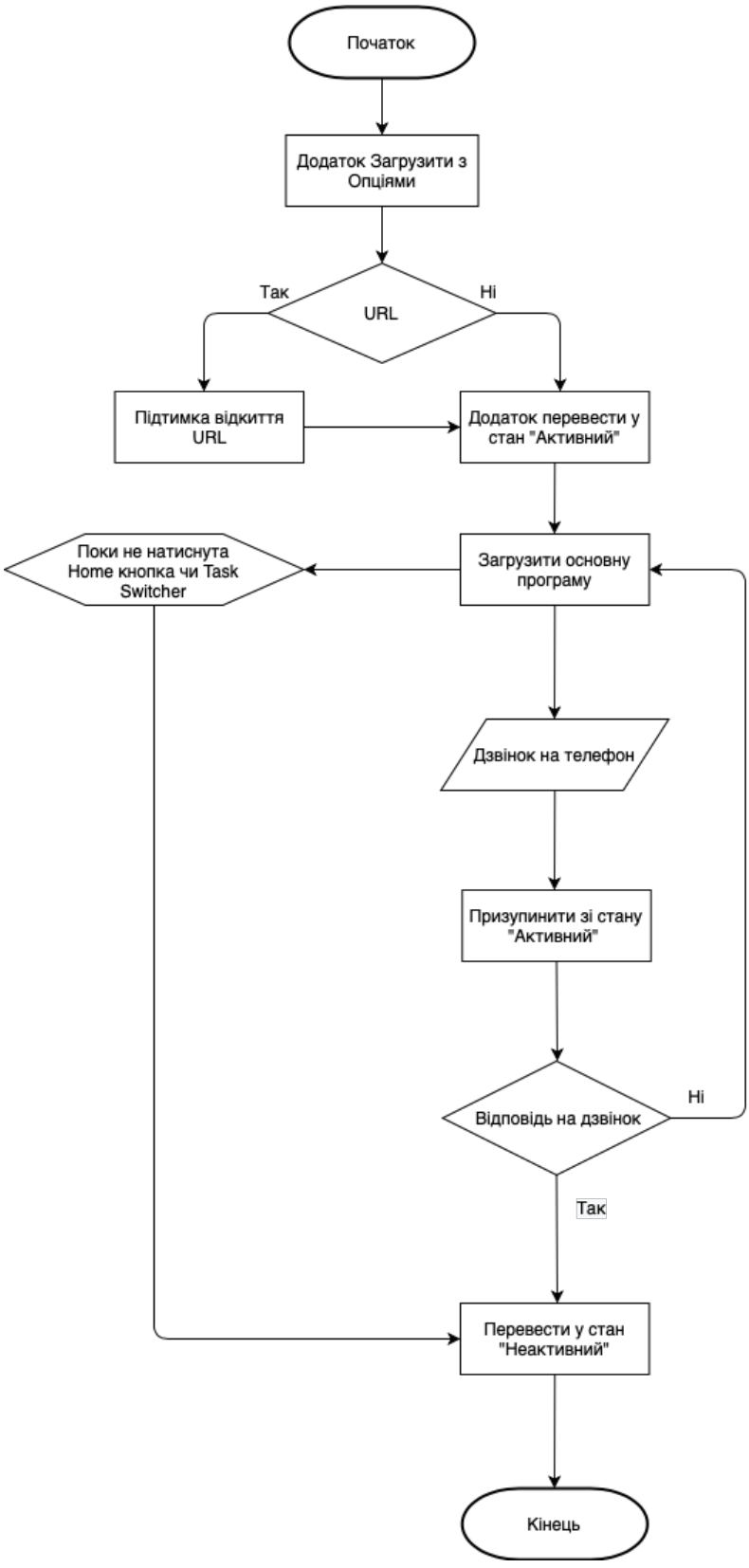
```
    }

    override func viewDidLoad() {
        super.viewDidLoad()
            configureEventsTemplates()
            setUpCollectionView()
            addMenuBar()
            numberOfItems = kpiTemplates.count
    }

    func emptyResults(results: [Searchable]){
        if let eventsCell = self.collectionView.cellForItem(at: IndexPath(row: 0, section: 0)) as?
EventTypeCollectionCell{
            eventsCell.events = []
            eventsCell.tableView.reloadData()
        }
          else if let peopleCell = self.collectionView.cellForItem(at: IndexPath(row: 1, section: 0))
as? PeopleTypeCollectionCell{
                peopleCell.people = []
                peopleCell.tableView.reloadData()
            }
    }

    func getResultsBySearch(with query : String, results: [Searchable]?, error: String?){
         DispatchQueue.main.async { [unowned self] in
            //print(query ,results, error,results.self)
        if results != nil {
            if results?.isEmpty == false {
            if results is [Event]{
                if let eventsCell = self.collectionView.cellForItem(at: IndexPath(row: 0, section: 0))
as? EventTypeCollectionCell{
                    eventsCell.events = results as! [Event]
                    eventsCell.tableView.reloadData()
                }
            }
            else if results is [User]{
                if let peopleCell = self.collectionView.cellForItem(at: IndexPath(row: 1, section: 0))
as? PeopleTypeCollectionCell{
                    peopleCell.people = results as! [User]
                    peopleCell.tableView.reloadData()
                }
            }

        }
            else{
                self.emptyResults(results: results!)
            }
        }
        else {print(query,error!)}
        }
    }
```

```
    func showSorryPage(){
        let alert = UIAlertController(title: "Temporarily Unavailable", message: "This feature is
temporarily unavailable. Try again later", preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
        self.present(alert, animated: true, completion: nil)
    }

    func configureEventsTemplates () {
        kpiTemplates = [KpiTemplate(isActive: true, titleName: "Hello", request: "http://d", index:
0), KpiTemplate(isActive: true, titleName: "Hello2", request: "http://d", index: 1)]
    }

    func setUpCollectionView(){
        collectionView.delegate = self
        collectionView.dataSource = self
        collectionView.zeroConstraint(with: self.view)


        if let layout = collectionView.collectionViewLayout as? UICollectionViewFlowLayout {
            layout.scrollDirection = .horizontal
            layout.minimumLineSpacing = 0
        }
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {
        return CGSize(width: view.frame.width, height: view.frame.height - 100)
    }


    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section:
Int) -> Int {
        return numberOfItems
    }

    func collectionView(_ collectionView: UICollectionView, willDisplay cell:
UICollectionViewCell, forItemAt indexPath: IndexPath) {
        print("cell ",indexPath.row)

        NotificationCenter.default.post(name: .sectionEntered, object: nil, userInfo:
["Section":indexPath.row])
    }



    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath:
IndexPath) -> UICollectionViewCell {

        var model : CellViewAnyModel
        currentSectionType = sectionTypes[indexPath.row]

        switch currentSectionType {

        case .event:
```

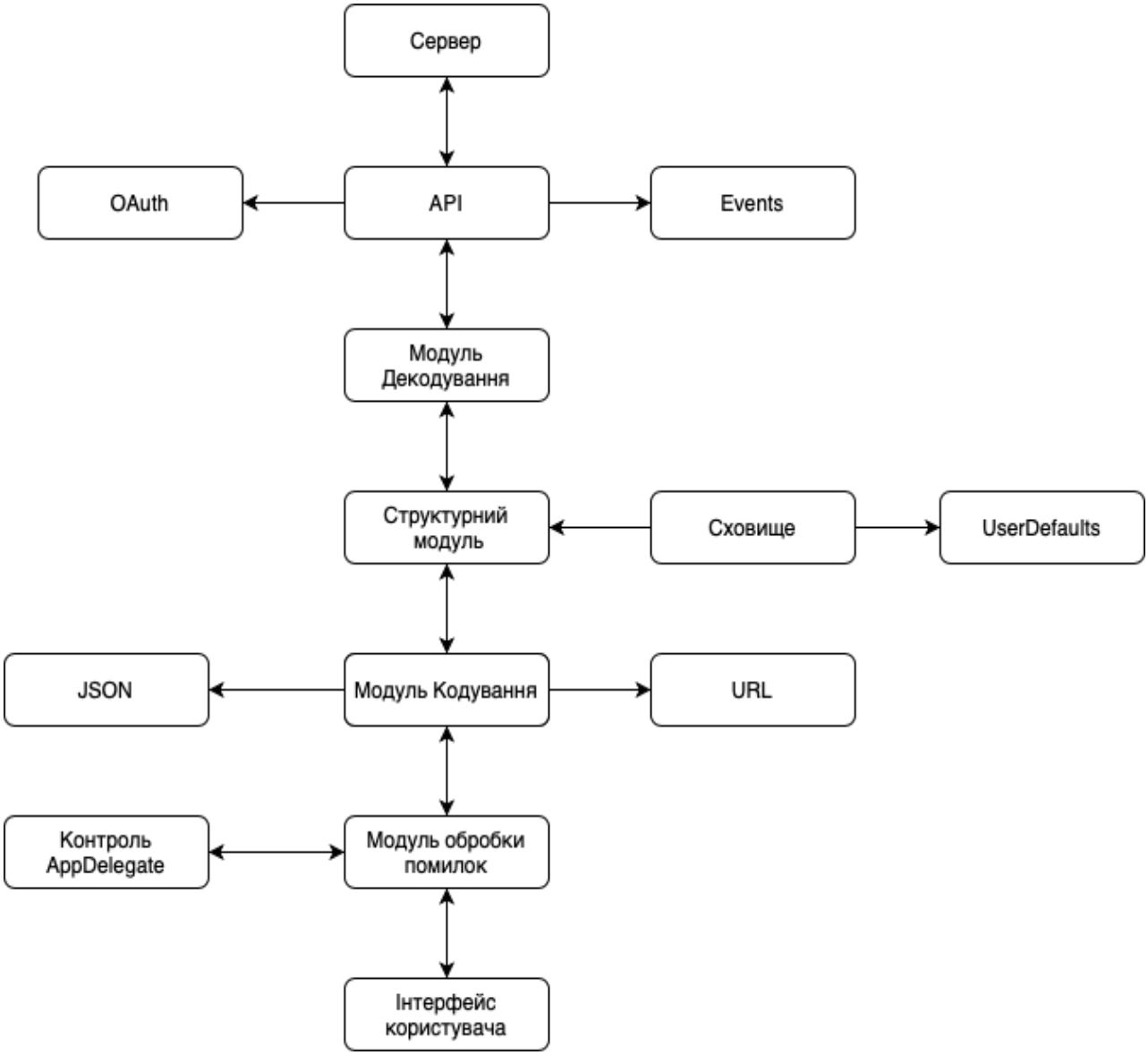
```
            model = EventCollectionCellModel()
        case .people:
            model = PersonCollectionCellModel()
        }
        return collectionView.dequeueReusableCell(withModel: model, for: indexPath)
    }

    func scrollViewDidScroll(_ scrollView: UIScrollView) {
        menuBar.horBarLeftConstrint?.constant = scrollView.contentOffset.x /
CGFloat(numberOfItems)
    }

    func scrollViewWillEndDragging(_ scrollView: UIScrollView, withVelocity velocity:
CGPoint, targetContentOffset: UnsafeMutablePointer<CGPoint>) {

        let index = Int(targetContentOffset.pointee.x / self.view.frame.width)
        let indexPath = IndexPath(row: index, section: 0)

        menuBar.collectionView.selectItem(at: indexPath, animated: true, scrollPosition: .left)
        setTitileForIndex(indexPath.item)
    }

    func scrollViewDidEndDecelerating(_ scrollView: UIScrollView) {
        switch collectionView.indexPath(for: collectionView.visibleCells.first!) {
        case IndexPath(row: 0, section: 0):
            currentSectionType = .event
        case IndexPath(row: 1, section: 0):
        currentSectionType = .people
        default:
            print("Unknown cell")
        }

    func scrollToMenuBarIndex(index: Int){
        let indexPath = IndexPath(row: index, section: 0)
        collectionView.scrollToItem(at: indexPath, at: .right , animated: true)
        setTitileForIndex(index)

    }
    private func setTitileForIndex(_ index: Int){
        if let titleLabel = navigationItem.titleView as? UILabel{
            titleLabel.text = "\(kpiTemplates[index].titleName)"
        }

    }


}
```



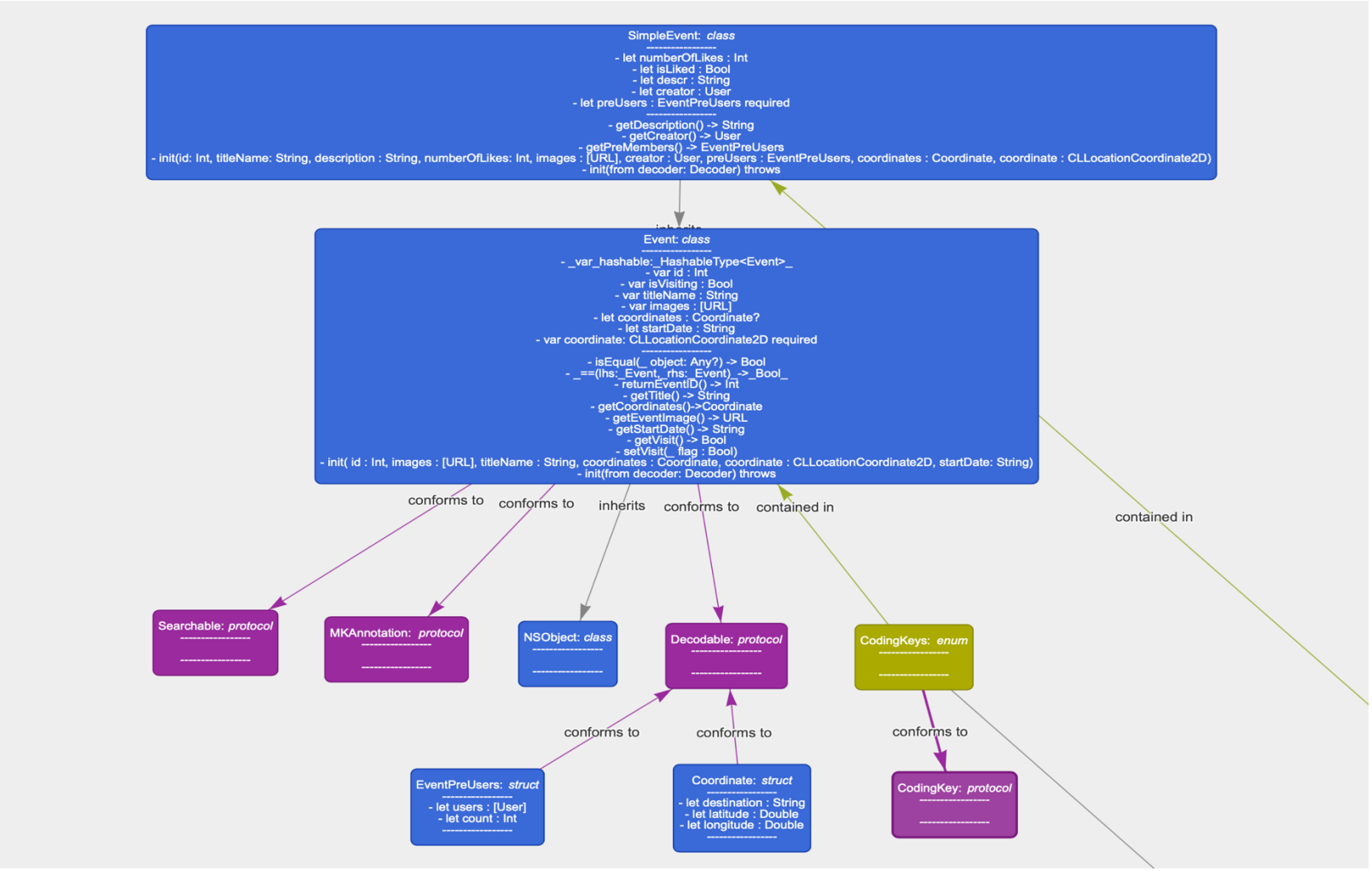
					ДП 6424. 00.004 Д1	
Зм.	*Арк*	*№ докум.*	*Підпис*	*Дата*		
		Соляніков П.О.			*Загальна схема додатку.*	*Літ.* / *Аркуш* / *Аркушів*
Перевірив.		*Каплунов А.І*				1 / 1
.						
Н. Контр.		*Сімоненко В.П*				*НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64*
Затвердив.						



					ДП 6424. 00.005 Д2		
Зм.	*Арк*	*№ докум.*	*Підпис*	*Дата*			
		Соляніков П.О.			*Принципова схема.*	*Літ.*	*Аркуш* 1 / *Аркушів* 1
Перевірив.		*Каплунов А.І*					
.						*НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64*	
Н. Контр.		*Сімоненко В.П*					
Затвердив.							



					ДП 6424. 00.006 Д3		
Зм.	Арк	№ докум.	Підпис	Дата			
		Соляніков П.О.			Структурна схема	Літ.	Аркуш 1 / Аркушів 1
Перевірив.		Каплунов А.І					
.						НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64	
Н. Контр.		Сімоненко В.П					
Затвердив.							



					ДП 6424. 00.007 Д4	
Зм.	Арк	№ докум.	Підпис	Дата		
		Соляніков П.О.			Діаграма класів сутності Подія	Літ. / Аркуш: 1 / Аркушів: 1
Перевірив.		Каплунов А.В.				
.						
Н. Контр.		Сімоненко В.П				НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64
Затвердив.						